I Know What Your Layers Did: Layer-wise Explainability of Deep Learning Side-channel Analysis

Guilherme Perin[1], Lichao Wu[2] and Stjepan Picek[1]

[1] Radboud University, The Netherlands
[2] Delft University of Technology, The Netherlands

Abstract. Masked cryptographic implementations can be vulnerable to higher-order attacks. For instance, deep neural networks have proven effective for second-order profiling side-channel attacks even in a black-box setting (no prior knowledge of masks and implementation details). While such attacks have been successful, no explanations were provided for understanding why a variety of deep neural networks can (or cannot) learn high-order leakages and what the limitations are. In other words, we lack the explainability of how neural network layers combine (or not) unknown and random secret shares, which is a necessary step to defeat, e.g., Boolean masking countermeasures.
In this paper, we use information-theoretic metrics to explain the internal activities of deep neural network layers. We propose a novel methodology for the explainability of deep learning-based profiling side-channel analysis to understand the processing of secret masks. Inspired by the Information Bottleneck theory, our explainability methodology uses perceived information to explain and detect the different phenomena that occur in deep neural networks, such as fitting, compression, and generalization. We provide experimental results on masked AES datasets showing where, what, and why deep neural networks learn relevant features from input trace sets while compressing irrelevant ones, including noise. This paper opens new perspectives for the understanding of the role of different neural network layers in profiling side-channel attacks.

Keywords: Side-channel Analysis · Deep learning · Countermeasures · Explainability · Mutual Information · Information Bottleneck Theory

1 Introduction

Side-channel attacks (SCA) represent powerful non-invasive attacks that exploit unintentional leakages of confidential information from electronic devices [MOP06]. During cryptographic executions, the devices leak information through different side channels, such as power consumption [KJJ99], electromagnetic emission [QS01], or execution time [Koc96]. Depending on the application, attacks revealing secret information can lead to serious consequences for the security industry [RLMI21]. Thus, chip manufacturers are interested in assessing the vulnerabilities of their designs before putting them on the market or into a certification process. To address this goal, security evaluators deploy different side-channel attacks that can be categorized as profiling (e.g., Template Attacks [CRR03]) and non-profiling attacks (e.g., Simple Power Analysis [Koc96] and Differential Power Analysis [KJJ99]).

Deep learning (DL-SCA) has drawn significant interest from researchers in the SCA domain [PPM+21]. Powerful attacks against protected cryptographic implementations have been demonstrated [MPP16, CDP17, KPH+19], especially against datasets containing first-order (Boolean) masked AES software or hardware [WHJ+21] implementations. Moreover, the SCA community constantly manages to improve the performance of deep learning. For example, the first publication using the ASCAD dataset with the fixed key required around 400 traces to break the target [BPS+20]. Today, we can break the same dataset with a single attack trace [PWP21]. Unfortunately, despite all of the advances in enhancing the attack efficiently, we still lack the knowledge in understanding *why* neural networks break a target.[1] Although past research put effort into the understanding of deep learning models [MDP20, HGG19, WWJ+21], there is still an evident lack of knowledge about, for instance, *how* deep neural networks learn to defeat masking countermeasures. We refer interested readers to several recognized challenges in deep learning-based SCA, especially the one connected to the explainability of the masking countermeasure processing [PPM+21].

We argue that the ability to explain why a machine learning model behaves a certain way is (at least) as important as improving the attack performance. With a deeper understanding of the neural network, an evaluator can: 1) mount more powerful attacks, 2) improve the security of devices, and 3) ultimately offer devices that are resilient against the strongest attacks. To conclude, we require **Ex**plainable **D**eep **L**earning-based **S**ide-**C**hannel **A**nalysis: **ExDL-SCA**. ExDL-SCA has two main goals: 1) explain where the leakage comes from and 2) explain how a profiling model defeats different countermeasures. To reach those goals, we propose the following questions to be answered with ExDL-SCA:

1. **Where**. By answering this question, we can explain the contribution of different layers in a neural network.
2. **What**. By answering this question, we can explain what a neural network does to break the target, i.e., how it defeats noise and countermeasures.
3. **Why**. By answering this question, we can explain why a neural network behaves in a certain way, e.g., breaking a target or failing to do so.

This paper proposes an explainability methodology that infers how much information the black-box model learns from secret masks. Secret masks are not supplied during the training of a black-box profiling model and are only considered for explainability. For that, we consider information-theoretic methods. The starting point of our explainability methodology is Information Bottleneck Theory (IB) [TPB00, ST17], which has sparked a lot of interest from deep learning community as a potential theoretical framework to explain how deep neural networks achieve enormous success in many different applications. In essence, IB theory suggests that deep neural network training undergoes two different phases: fitting and compression. The fitting phase is supposedly fast and is characterized by hidden layers trying to maximize information about $\mathcal{X}$ while compression is slower, and it is responsible for the generalization ability of the model. According to [ST17], during the fitting phase, the model already shows generalization, which is enhanced during the compression phase. When and how these phases happen in a specific model depends on the model architecture, hyperparameters, and the dataset's characteristics. The compression phase is particularly important, as in this phase, the network starts to compress noise and other irrelevant features while it preserves only relevant features from input training data $\mathcal{X}$. For that, the IB theory requires the computation of mutual information between (usually) high dimensional input data $\mathcal{X}$ (such as side-channel measurements) and (potentially) high dimensional intermediate network representations T (the output of a hidden layer), i.e., $I(\mathcal{X}, T)$. However, as we will explain in this paper, computing $I(\mathcal{X}, T)$ is particularly hard for discrete and high dimensional representations [GP20, SBD+18], which limits the estimation of the compression phase during training. As a solution, we adapt the IB framework to Perceived Information [BHM+19] metric, which allows us to precisely

[1]Deep neural networks usually have complex architectures that are difficult to interpret and explain. By interpretable machine learning, we consider designing machine learning models that are inherently interpretable or answering the question of how the model work [MV20]. By explainability (explainable AI - XAI), we consider how to provide post hoc explanations of the black-box models.

explain fitting, compression, and generalization phases in different hidden layers. Thus, we verify, during training, **where**, **what**, and **why** every neural network layer actually learns from high-order leakages. Thus, our explainability methodology allows security evaluators to verify, with more specific information from hidden layers, what a profiling model learns (or not) from the implemented countermeasures. Our main contributions are:

1. We discuss explainability in the context of DL-SCA and recognize the three questions that need to be answered for explainable DL-SCA.
2. We define a new methodology to infer the information learned by hidden neural network layers during profiling. The procedure we follow is based on an autoencoder idea, where we extract an encoded version of the input datasets (profiling and attack sets) from each hidden layer. Then, a simple multilayer perceptron classifier (which is not the same one used for profiling) is trained with the encoded profiling set. Next, we predict the encoded attack set on this simple classifier and compute the perceived information to quantify how much information a hidden layer has about a specific feature, like a secret share in a Boolean masking scheme. This process is repeated for all hidden layers. Our explainability methodology is depicted in Figure 5. Our method allows an evaluator to measure how the input information leakage is learned and conveyed layer by layer in a deep neural network. Furthermore, our method can show in what layer the information bottleneck is inherently implemented in order to compress irrelevant input information (such as noise) and to preserve relevant leakages to break masking countermeasures.
3. We provide experimental results on publicly available datasets and different neural network architectures. All our results indicate that the information bottleneck theory is a valid method to explain the different phases of deep neural network training. Then, we apply our explainability methodology to a combination of masking and desynchronization countermeasures, showing our approach to work even if different countermeasures are used.
4. Finally, we propose a number of use cases where our explainability approach can be used to improve the performance of deep learning in SCA.

This paper is organized as follows. We start by providing background information in Section 2 while related works are discussed in Section 3. Section 4 introduces our novel explainability methodology, and Section 5 discusses the compression in deep networks with practical examples on different datasets. Experimental results with different protected AES datasets and neural network architectures are provided in Section 6. Our explainability methodology opens a new perspective to understanding the role of different hidden deep neural network layers. Thus, we also provide a list of possible use cases for our proposed approach in Section 7. Finally, conclusions and future work directions are provided in Section 8.

2 Background

2.1 Notations and Terminology

We refer to $\mathcal{X}$ as a set of side-channel measurements and $\mathtt{x}_i$ is the i-th observation of $\mathcal{X}$. $\mathcal{X}_p$ is a set of profiling side-channel measurements and $\mathcal{X}_a$ is the attack set with lengths n_p and n_a, respectively. Each side-channel measurement $\mathtt{x}_i$ represents the side-channel leakages of a cryptographic operation having input data $\mathtt{d}_i$ and encryption key $\mathtt{k}_i$.[2] We refer to $\mathcal{Y}$ as the set of hypothetical leakage values (or labels) for $\mathcal{X}$ where $\mathtt{y}_i = f(\mathtt{d}_i, \mathtt{k}_i)$ is one element of $\mathcal{Y}$, and f denotes a leakage selection function (i.e., f can be represented by an `S-box`

[2] Instead of the encryption function and plaintext, it is also possible to consider decryption function and ciphertext, but for simplicity, we consider encryption only.

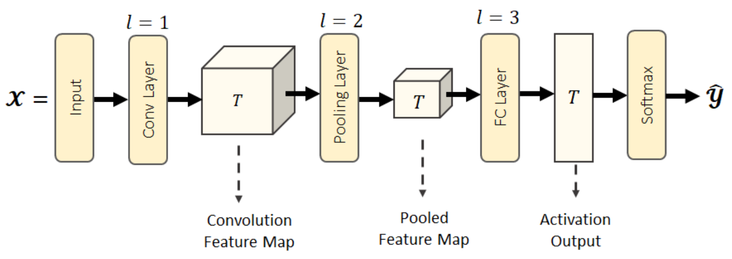


Figure 1: Neural network (CNN) intermediate representations.

operation in the first encryption AES round). In the case of masking countermeasures, $\mathtt{m}_r$ refers to (r)-th secret share byte while $\mathcal{Y}_{m(r)}$ refers to the set of mask shares in a dataset.

With respect to information-theoretic notions, we refer to $\mathtt{p}(\mathtt{x}_i)$ as the probability of observing $\mathtt{x}_i$ and $\mathtt{p}(\mathtt{y}_i|\mathtt{x}_i)$ as the probability of observing $\mathtt{y}_i$ given $\mathtt{x}_i$. $H(\mathcal{X})$ is the entropy of $\mathcal{X}$ while $H(\mathcal{Y}|\mathcal{X})$ gives the conditional entropy of $\mathcal{Y}$ given $\mathcal{X}$. The mutual information between $\mathcal{X}$ and $\mathcal{Y}$ is given by $I(\mathcal{X};\mathcal{Y})$.

For neural network representations, we refer to T as an encoding providing an intermediate representation of $\mathcal{X}$ in a neural network (e.g., T could represent the feature map output of a convolution layer or the activations output of a fully-connected layer) and $\mathtt{t}_i$ is an observation of T. The term L refers to the number of hidden layers (excluding the output layer from the counting). The index of a hidden layer is given by l. The term X^l indicates the predicted output of a hidden layer l when the input data to the network is $\mathcal{X}$. Finally, $\widehat{\mathcal{Y}}$ is the output prediction from a neural network. Figure 1 provides an example of a convolutional neural network with representations.

In this paper, the term *sample* refers to a point of interest $\mathtt{x}_i[j]$ in a side-channel measurement $\mathtt{x}_i$. The term *feature* refers to the meaning of some information contained in $\mathcal{X}$. For example, when $\mathcal{X}$ represents the set of side-channel measurements from the AES encryption process, the leakage of an intermediate byte in each measurement $\mathtt{x}_i$, which is given by a label set $\mathcal{Y}$, is a feature of $\mathcal{X}$.

We also define specific notations for neural networks. Convolutional neural networks have layer-wise structure according to the Eq. (1), where $C(\mathtt{fi},\mathtt{ks},\mathtt{st})$ denotes a convolution layer with $\mathtt{fi}$ filters, kernel size $\mathtt{ks}$, and stride $\mathtt{st}$, A is activation layer (which can be RE in case of $\mathtt{relu}$, SE in case of $\mathtt{selu}$, or simply E in case of $\mathtt{elu}$), BN is a batch normalization layer, $AP(\mathtt{ps},\mathtt{st})$ is an average pooling layer with pooling size $\mathtt{ps}$ and stride $\mathtt{st}$, $FC(\mathtt{ne})$ is a fully-connected layer with $\mathtt{ne}$ neurons and $S(\mathtt{c})$ is a Softmax layer with $\mathtt{c}$ output neurons. The superscripts n_c and n_{fc} indicate the number of convolution blocks and fully-connected layers, respectively.

$$\mathcal{X} \rightarrow [C(\mathtt{fi},\mathtt{ks},\mathtt{st}) \rightarrow A \rightarrow BN \rightarrow AP(\mathtt{ps},\mathtt{st})]^{n_c} \rightarrow [FC(\mathtt{ne}) \rightarrow A]^{n_{fc}} \rightarrow S(\mathtt{c}) \rightarrow \widehat{\mathcal{Y}}. \tag{1}$$

Similarly, a multilayer perceptron (MLP) is defined according to the following layer-wise notation:

$$\mathcal{X} \rightarrow [FC(\mathtt{ne}) \rightarrow A]^{n_{fc}} \rightarrow S(\mathtt{c}) \rightarrow \widehat{\mathcal{Y}}. \tag{2}$$

2.2 Deep Learning-based Profiling SCA against Masked Implementations

In classification applications, a neural network model represents a function that maps input data $\mathcal{X}$ into a finite number of output class probabilities $\hat{\mathcal{Y}}$. The mapping is performed by a learned function $f(\mathcal{X},\theta) \rightarrow \widehat{\mathcal{Y}}$, where θ is a set of parameters learned during the training phase by minimizing a loss function.[3] The learned mapping between input side-channel traces $\mathcal{X}$ and outputs probabilities $\hat{\mathcal{Y}}$ depends on the estimated number of classes presented in $\mathcal{X}$. This number of classes, $|\mathcal{Y}|$, is derived from a leakage function that indicates the hypothetical leakage value in a side-channel measurement.

To protect against side-channel attacks, masking is implemented to break the statistical dependence between side-channel measurements (e.g., power consumption) and hypothetical leakage values. For an m-order masking scheme, an intermediate byte b in a cryptographic algorithm is protected as follows:

$$\mathtt{b}_m = \mathtt{b} \diamond \mathtt{m}_1 \diamond \mathtt{m}_2 \cdots \diamond \mathtt{m}_m, \tag{3}$$

where $\diamond$ can indicate a Boolean [CJRR99], arithmetic [GPQ11], multiplicative [GPQ11], or affine [FMPR10] operation.

The leakage function $g(\cdot) = \mathcal{L}$ defined for a second-order profiling SCA is supposed to learn how to combine two unknown variables $\mathtt{m}_1$ and $\mathtt{m}_2$ according to:

$$\mathcal{L} = g(\mathtt{m}_1, \mathtt{m}_2) = \mathtt{m}_1 \diamond \mathtt{m}_2, \tag{4}$$

where g is a function mathematically combining two variables through operation $\diamond$. In our analysis, $\mathtt{m}_1$ will always be given by an 8-bit mask share randomly generated for each encryption execution while $\mathtt{m}_2$ will be given by an 8-bit masked `S-box` output of the first AES encryption round, i.e., $\mathtt{m}_2 = \texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1$. For instance, the leakage function for a second-order attack on a masked AES implementation is defined as:

$$\mathcal{L} = g(\mathtt{m}_1, \texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1) = \mathtt{m}_1 \oplus \texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1 = \texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i), \tag{5}$$

where $\diamond = \oplus$. A side-channel measurement $\mathtt{x}_i$ containing second-order leakages must embed leakage of information from the treatment of masked `S-box` output $(\texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1)$ and, at least, the loading of mask share $\mathtt{m}_1$ from memory. We can finally assume that to implement second-order profiling, a neural network must learn the following mapping:

$$f(\mathcal{X}, \mathcal{L}, \theta) = f(\mathcal{X}, g(\texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1, \mathtt{m}_1), \theta) \rightarrow \hat{\mathcal{Y}}. \tag{6}$$

This means that a neural network can learn a mapping from side-channel traces $\mathcal{X}$ to output class probabilities $\hat{\mathcal{Y}}$ that represents (ideally) the *xor* between two random 8-bit variables $\texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1$ and $\mathtt{m}_1$. In essence, the leakage function represented by learned parameters θ defines how continuous variables or input features $(\mathcal{X})$ (i.e., raw or pre-processed trace samples) are converted into hypothetical discrete leakage values $g(\mathcal{X}) \xrightarrow{\mathcal{L}} \hat{\mathcal{Y}}$, where $\hat{\mathcal{Y}}$ can also be seen as the set of predicted labels. As a consequence, a neural network that can learn second-order leakages defines a mapping with an intermediate function that can be given by one or more hidden layers, which learn how to implement $g(\texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1, \mathtt{m}_1)$.

The trained neural network, therefore, implements the following path:

$$\mathcal{X} \rightarrow T_1 \rightarrow \cdots \rightarrow T_L \xrightarrow{Softmax} \hat{\mathcal{Y}} \equiv \mathcal{X} \rightarrow g(\texttt{S-box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1, \mathtt{m}_1) \xrightarrow{Softmax} \hat{\mathcal{Y}}. \tag{7}$$

From Eq. (7), we can immediately verify that:

$$T_1 \rightarrow \cdots \rightarrow T_L \equiv g(\mathtt{m}_1, \mathtt{m}_2). \tag{8}$$

A loss function assesses the overall variation between expected (ground truth) hypothetical leakages (or labels) $\mathcal{Y}$ and predicted hypothetical leakages $\hat{\mathcal{Y}}$. Common hypothetical leakage models for side-channel analysis include Identity, Hamming weight, Hamming

[3]In this paper, we always consider categorical cross-entropy as the loss function. We emphasize that categorical cross-entropy is commonly used in deep learning-based SCA.

distance, or simply bit-level models (e.g., the least or most significant bits). Besides the predefined number of classes for classification, the trained neural network has no other information on converting input features into labels. Therefore, training a model inherently assumes that the network will automatically learn the leakage model properties by implementing the mapping from Eq. (6).

Following the same principle, for a third-order neural network-based profiling SCA against a second-order masking scheme, the leakage function that the trained model is expected to learn is given by:

$$\mathcal{L} = g(\mathtt{m}_1, \mathtt{m}_2, \mathtt{m}_3) = \mathtt{m}_1 \diamond \mathtt{m}_2 \diamond \mathtt{m}_3 = (\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \diamond \mathtt{m}_1 \diamond \mathtt{m}_2) \diamond \mathtt{m}_1 \diamond \mathtt{m}_2 = \mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i). \quad (9)$$

In this paper, we provide empirical experiments to analyze the performance of deep neural networks against a first-order Boolean masking scheme and leave the analysis of high-order masking, including other masking schemes, for future work. Nevertheless, the explainability methodology proposed in Section 2.4 applies to different masking schemes.

2.3 Information-theoretic Concepts

Different metrics are used to compare different profiling models. Guessing entropy [Mas94] and success rate [SMY09] became mainstream approaches to estimate the performance of a model in recovering the secret with a certain number of measurements. In a deep learning context, alternative metrics and frameworks can provide more insights into attack performance. In this section, we review the two main information-theoretic approaches that are used in this paper. The first, Perceived Information [BHM+19], was already proven to be an optimization goal in deep learning-based profiling attacks [MDP20], in which maximizing this metric is the same as minimizing the cross-entropy loss function. The second approach is Information Bottleneck Theory (IB) [TPB00], which provides a framework to quantify the information present in internal deep neural network representations T.

2.3.1 Perceived Information

In [BHM+19], the authors discussed Perceived Information (PI) as the information metric to measure the attack complexity. PI measures how much information a model can obtain from the test side-channel measurements and indicates the complexity of an attack in terms of the number of measurements to recover the secret. The PI calculation is provided as:

$$\widehat{PI}(\mathcal{X}, \mathcal{Y}) = H(\mathcal{Y}) + \sum_{j=1}^{|\mathcal{Y}|} p(\mathtt{y} = j) \frac{1}{n_{y=j}} \sum_{i=1}^{n_{y=j}} \log_2 \hat{p}(\mathtt{y} = j | \mathtt{x}_i^{y=j}), \quad (10)$$

where $H(\mathcal{Y})$ is the entropy of $\mathcal{Y}$, $\hat{p}(\mathtt{y} = j|\mathtt{x}_i^{y=j})$ is the probability of a model to predict a trace $\mathtt{x}_i^{y=j}$ labeled as $y = j$ and $n_{y=j}$ is the number of attack traces labeled as $y = j$ for key candidate $\mathbf{k}$.

As our analysis follows a known-key setting, the correct key from the attack set is always known, and PI values are always computed for the correct key candidate $\mathbf{k}^*$. The PI calculation allows the evaluator to estimate the minimum number of required attack traces $\hat{N}_{SR,PI}$ to recover the key:

$$\hat{N}_{SR,PI} \geq \frac{c(SR)}{\widehat{PI}(\mathcal{X}, \mathcal{Y})}, \quad (11)$$

where $c(SR)$ is a small constant related to the expected success rate $SR \in [0, 1]$. For instance, when targeting intermediate variables processed by a n bit devices, $c(SR)$ is given by [dCGRP19]:

$$c(SR) = n - (1 - SR). \log_2(2^n - 1) + SR \log_2(SR) + (1 - SR) \log_2(1 - SR). \quad (12)$$

2.3.2 Information Bottleneck Theory

The proposed explainability methodology in Section 2.4 is based on the Information Bottleneck [TPB00] theory. This theory suggests that learning from input $\mathcal{X}$ is a task-specific process undergoing (as much as possible) compression of $\mathcal{X}$ into an intermediate representation (or encoding) T that keeps relevant information to generalize to $\mathcal{Y}$. In other words, the representation T acts like a bottleneck that "squeezes" the relevant information that $\mathcal{X}$ contains about $\mathcal{Y}$, hence the name "information bottleneck". This relation is also represented by the following Lagrange multiplier, also known as the information bottleneck cost function:

$$\mathcal{L}_{IB} = I(\mathcal{X};T) - \beta I(T;\mathcal{Y}), \tag{13}$$

where $\beta \geq 0$ is the Lagrangian multiplier attached to the constrained relevant information. β controls the trade-off between compression and preserving information about $\mathcal{X}$. When $\mathcal{L}_{IB}$ is considered as the loss function (which is not the case in our analyses), the goal is to minimize $\mathcal{L}_{IB}$ for a correctly selected β. Here, the main goal is to learn an encoding T that is maximally expressive about $\mathcal{Y}$ while being maximally compressive about $\mathcal{X}$. Intuitively, the term in $I(T;\mathcal{Y})$ suggests that T should be as predictive as possible about $\mathcal{Y}$ and, for that, the $I(\mathcal{X};T)$ suggests that T should "forget" information about $\mathcal{X}$.

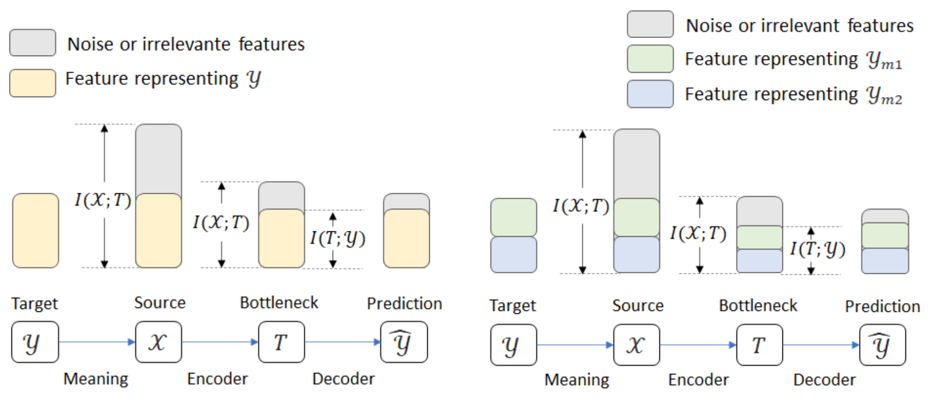
In [TPB00], the authors suggested that objective $\mathcal{L}_{IB}$ forces T to act like a minimal sufficient statistic of $\mathcal{X}$ for predicting $\mathcal{Y}$. This way, IB defines the curve indicating the maximum mutual information $I(T;\mathcal{Y})$ that can be achieved for minimum mutual information $I(\mathcal{X};T)$. This theory defines an upper bound for the maximum information that can be transferred to $\mathcal{Y}$ given a compressed representation of $\mathcal{X}$, which is T. Figure 2 illustrates the information bottleneck principle when the target $\mathcal{Y}$ is represented by one feature (Figure 2a) or two features (Figure 2b). Note that the input data $\mathcal{X}$ contains information about the feature(s) representing $\mathcal{Y}$ and also noise and irrelevant features. This input data is encoded, with loss of information, into a representation T containing information about the feature but can ideally discard noise. Therefore, T becomes the bottleneck.

From the IB theory [ST17], the authors proposed to visualize the deep learning optimization through an information plane that illustrates the course of learning with fitting, compression, and generalization phases. The fitting phase is the first phase that happens in training, which is characterized by the fast increase of $I(\mathcal{X};T)$ and $I(T;\mathcal{Y})$, while the second phase, compression, is responsible for enhancing model generalization and provides a reduction of $I(\mathcal{X};T)$ and a slow increase of $I(T;\mathcal{Y})$, in case the model shows no overfitting. When overfitting happens, $I(T;\mathcal{Y})$ should decrease while $I(\mathcal{X};T)$ may show no compression.

For deep learning-based SCA against masked implementations, we are interested in explaining how neural network hidden layers fit the secret shares, compress them into minimum sufficient statistics, and generalize to $\mathcal{Y}$ during training. For that, we will first adapt the IB principle to the perceived information, which is a side-channel metric capable of estimating attack complexity. As will be described in Section 4, the adaption to perceived information comes as a solution to the difficulty of computing mutual information $I(\mathcal{X};T)$ when $\mathcal{X}$ and T are high dimensional data. As we show in experimental results, the resulting IB-inspired explainability methodology can inform about different phenomena that occur in different hidden layers.

2.4 Interpretability and Explainability in Profiling SCA

The more complex the learning algorithm, it is potentially less interpretable and explainable. Recent publications in the deep learning-based profiling SCA domain have dominantly focused on finding efficient deep neural network architectures for a wide range of scenarios, see, e.g., [ZBHV19, RWPP21, WPP20]. However, the lack of understanding on *how* a deep



(a) One feature. **(b)** Two features.

Figure 2: Information bottleneck principle representation (illustration based on [Gut22].) As an analogy to SCA, the one-feature case is the process implemented when side-channel measurements $\mathcal{X}$ contain first-order leakages. The two-features case illustrates the case when $\mathcal{X}$ contains second-order leakages from features $\mathcal{Y}_{m1}$ and $\mathcal{Y}_{m2}$, which represent two secret shares.

learning model reaches a certain (lack of) generalization capacity makes those approaches less transferable to different attack settings. Indeed, an explainable deep learning model helps to conclude two main aspects of a security evaluation: (1) the security assurance of the target and (2) how assertive is the implemented profiling model. We list the following mechanisms for interpretability and explainability in profiling SCA:

Input gradients analysis [MDP19] This method implements the so-called sensitivity analysis of loss function concerning input features or side-channel samples:

$$I_g[j] = \frac{\partial F(\theta)}{\partial x_i}[j], \tag{14}$$

where j denotes the index of the j-th feature or sample in a i-th side-channel measurement x_i.

Layer-wise Relevance Propagation [HGG19] This method provides a way to assign a relevance magnitude to each neuron in a layer. The process starts from the last hidden layer and is computed layer-wise backward until the input layer is achieved. Like input gradient analysis, this technique indicates the most relevant points of interest for specific deep neural network decisions.

Ablation/"Surgery" technique [WWJ$^+$21] The main purpose of this approach is to understand the role of each neural network element (e.g., hidden layers or neurons and filter in more detailed analyses) concerning the bypassing of hiding countermeasures such as trace desynchronization and noise. By removing (i.e., ablating) elements of the network, it is possible to compare the profiling model performance before and after the presence of this element and the countermeasure. If the neural network works worse after removing a certain element, this indicates that the element has an important role in treating these hiding countermeasures. The study of ablation techniques requires the possibility of enabling and disabling hiding countermeasures. Hiding countermeasures can be simulated after side-channel acquisition. However, the same process is not possible with masking countermeasures, as one cannot simulate their absence after acquisition for the same processed intermediate variable.

2.5 Datasets

For our experimental results, we select, among several publicly available datasets, two masked AES datasets. We decided to consider the trace sets from the ASCAD and DPAv4 databases, whose keys in profiling and attack phases are different, and mask shares are also provided in the metadata. Additionally, we simulate the effect of a hiding countermeasure (desynchronization) on those datasets.

In security evaluations, different adversary assumptions are made, which result in different threat models. When we assume that an adversary has access to source code and/or internal target randomness (e.g., masks), the attacked samples usually correspond to the highest signal-to-noise ratio (SNR) peaks of leakages or to a trimmed interval where the main leakage is located. On the other hand, when assumptions about the strengths of an adversary are relaxed, the attacked interval is wider and includes a large number of samples that represent noise to the profiling model. In our case, for the evaluated AES datasets, we select attacking intervals where the main leakage for one target key byte is located but also includes leakages from other key bytes, as detailed below.

`ASCADr` This dataset contains 300 000 traces collected from a software implementation of AES 128,[4] where the first 200 000 measurements have random keys and are considered for profiling while 100 000 measurements contain a fixed key and, from this second set, we consider 5 000 for the attack phase. Each measurement contains 250 000 samples. This dataset was collected from an AES128 implementation featuring a first-order Boolean masking countermeasure. In previous works, a trimmed version of this dataset containing measurements with 1 400 samples is commonly adopted, which contains second-order leakages related to the third key byte only. For our experiments, we start from raw measurements containing 250 000 samples and select the interval from sample 70 000 until sample 90 000. Then, we apply a window resampling with a resampling window of 20 samples and a step of 10 samples, resulting in traces with 2 000 samples. These trimmed and resampled measurements contain second-order leakages related to the third key byte in the first encryption round but also include leakages from other key bytes.

`DPAv4.2` The `DPAv4.2` dataset contains side-channel measurements obtained from a masked AES 128 software implementation.[5] The countermeasure is based on RSM (*Rotation `S-box` Masking*). The original `DPAv4.2` contains 80 000 traces subdivided into 16 groups of 5 000 traces. Each group is defined with a separate and fixed key. Each measurement has 1 704 046 samples. In this work, we conduct our analyses on an interval resulting from the concatenation of two intervals from the original dataset. The main idea is to combine two trace intervals containing second-order leakages from several key bytes, including the attacked one. The second-order leakages include masked `S-box` output bytes and the corresponding masks. The first interval ranges from sample 265 000 until sample 280 000, while the second interval starts at sample 305 000 and finishes at sample 315 000. Thus, concatenating these two intervals results in measurements with 25 000 samples. We apply a resampling process with a resampling window of 10 and step of 5 to the concatenated intervals, resulting in 5 000 samples per measurement.

For both datasets, we consider the Identity leakage model for an output byte from the first AES encryption round, i.e., $\texttt{S-box}(\texttt{d}_i \oplus \texttt{k}_i)$.

[4] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key
[5] http://www.dpacontest.org/v4/42_doc.php

3 Related Works

Explainable AI (XAI) and explainable machine learning XML) are very active research domains, see, e.g., [BH21, GSC+19, BP21, LPK20, PDN22, Hol18]. Still, despite all the developments and various techniques proposed over the years, there is still no widely accepted approach that allows interpretability for diverse machine learning tasks. Interpretability and explainability in deep learning profiling SCA have received little attention in recent years. A larger focus has been put on neural network optimization to solve the difficult task of hyperparameter tuning. Still, the efforts to build various methodologies could be considered interpretability research since the authors (tried to) provide guidelines to build good neural networks, which intuitively means they could interpret what models do [ZBHV19, WAGP20].

In the SCA context, IB theory was indirectly considered in [CLM20] where the authors implemented leakage assessment with MINE (mutual information neural estimation) [BBR+18], a technique adapted from the IB principle. Moreover, in [PBP21], the authors implemented an early stopping mechanism for a deep learning-based profiling attack by maximizing $I(T;\mathcal{Y})$ for the output layer. In this case, the authors ignored $I(\mathcal{X};T)$ and $I(T;\mathcal{Y})$ for hidden layers and only focused on the output layer.

The more "direct" attempts at interpretability and explainability can be divided into approaches that concentrate on the input layer and the approaches that concentrate on the inner (hidden) layers. The techniques that concentrate on the input layer try to recognize the most important features (or, what is the influence of each feature on the performance of a neural network). Visualization techniques were the first attempt to explain what side-channel trace features have more impact on neural network decisions. Masure et al. [MDP19] provided visualization results through input gradient from the input network layer, and the authors verified that neural networks automatically detect the time location of secret shares even in the presence of desynchronization countermeasures. In [HGG19], the authors compared different visualization techniques in profiling SCA and considered them as side-channel attack distinguishers. However, such approaches cannot inform about internal representations from hidden layers.

To interpret the behavior of hidden layers in profiling SCA, the authors of [vdVPB20] considered Singular Vector Canonical Correlation Analysis (SVCCA) to explain what neural network layers learn from different side-channel traces. Still, the authors only managed to reach interpretability on a coarse level as even diverse datasets (side-channel dataset and image dataset) had more similarity than two side-channel datasets. Wu et al. [WWJ+21] proposed the adoption of ablation techniques to explain how different neural network configurations perform in the presence of different hiding countermeasures. While these results are very interesting, we note that they cannot explain the processing of masks, and the approach is rather involved and gives results that are (potentially) difficult to interpret.
Despite the progress obtained in the last few years, what happens in each hidden layer when fitting higher-order leakages and how (if) the countermeasures are defeated are still open questions.

4 Explainability Methodology for Profiling SCA

In this section, we describe the proposed explainability methodology. The process allows us to understand (and explain) how secret mask shares are fit by hidden layers in a deep neural network. The proposed solution works by adding extra calculations during deep neural network training. The model is trained with profiling traces $\mathcal{X}_p$ labeled in a black-box way with $\mathcal{Y}_p$. At the end of each training epoch, the model predicts profiling and attack traces $\mathcal{X}_p$ and $\mathcal{X}_a$, respectively. The output of each hidden layer l is saved as encoded versions of profiling and attack sets, i.e., $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$, respectively. The shapes of $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$ depends on the output layer dimensions. Basically, $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$ are activation outputs from a hidden layer l (see Figure 3).

The proposed explainability methodology evaluates the amount of information that $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$ contain from secret shares labels $\mathcal{Y}_{m1}$ and $\mathcal{Y}_{m2}$. Thus, we first propose (Section 4.1) an adaptation from mutual information to perceived information as a way to measure the amount of information that an intermediate network representation (in case $\mathcal{X}_p^l$ or $\mathcal{X}_a^l$) contains about features (e.g., secret shares) embedded in input data. To simplify, we elaborate our methodology based on a first-order Boolean masking scheme in which a neural network needs to learn from two secret shares $\mathtt{m}_1$ and $\mathtt{m}_2$, represented by features or label sets $\mathcal{Y}_{m1}$ and $\mathcal{Y}_{m2}$. However, this explainability methodology can infer the amount of information from any type of feature, including first-order leakages from unprotected devices.

4.1 Measuring Information Learned by Hidden Layers with Perceived Information

We propose an alternative solution to measure the amount of information a hidden layer learns from input data. More specifically, we are interested in a metric that indicates the minimum amount of information we can get from secret share labels $\mathcal{Y}_{m1}$ and $\mathcal{Y}_{m2}$ (and, eventually, true labels $\mathcal{Y}$) once we obtain intermediate network representations $\mathcal{X}_p^l$ or $\mathcal{X}_a^l$. Before developing our approach, we first explain why we cannot directly consider the method proposed in [ST17] to compute mutual information $I(\mathcal{X};T)$ between input training data $\mathcal{X}$ and an intermediate network representation T.

Directly applying MI estimation to compute $I(\mathcal{X};T)$ according to the original proposition [ST17] requires to compute mutual information between two high dimensional data $\mathcal{X}$ (side-channel traces) and T (layer representation of T). This way, an accurate estimation of MI requires exponentially more data, especially for histogram-based mutual estimation, as will be explained next. Since mutual information is symmetric, $H(\mathcal{X}) - H(\mathcal{X}|T) = H(T) - H(T|\mathcal{X})$, $I(\mathcal{X};T)$ can be computed according to:

$$I(\mathcal{X};T) = H(T) - H(T|\mathcal{X}) = H(T) - \sum_{i=1}^{n_p} \mathtt{p}(\mathtt{x}_i) \sum_{j=1}^{n_p} \mathtt{p}(\mathtt{t}_j|\mathtt{x}_i) \log_2 \mathtt{p}(\mathtt{t}_j|\mathtt{x}_i). \tag{15}$$

As discussed in [SBD+18], the histogram-based estimation of mutual information requires a correct selection of the number of bins. When the number of bins is too large to keep the precision of T, every input $\mathtt{x}_i$ yields a different activation pattern $\mathtt{t}_j$ in each hidden layer. In other words, due to the high dimensionality of $\mathcal{X}$ and T, it is often impossible to have two input traces $\mathtt{x}_i$ that generate two identical intermediate network representations $\mathtt{t}_i$. This will result in $H(T|\mathcal{X}) = 0$ because the conditional probabilities $\mathbf{p}(\mathtt{t}_j|\mathtt{x}_i)$ become 1 for all $\mathtt{x}_i$ and all $\mathtt{t}_j$ and, consequently, $I(\mathcal{X};T) = H(T)$. This result would wrongly indicate the compression of input $\mathcal{X}$ during training, as we would only be computing the entropy of T. A possible solution to obtain $H(T|\mathcal{X}) > 0$ is to select smaller number of bins. Nevertheless, this would add too much noise to the mutual information calculation, also leading to wrong estimations (e.g., see Appendix A in [PBP21]). Some works from the deep learning community evaluated the difficulty of computing $I(\mathcal{X};T)$ as a way to more correctly estimate the compression of $\mathcal{X}$ into T. In [SS17], the authors proposed the deterministic information bottleneck as an alternative cost function for Eq. (13) by replacing $I(\mathcal{X};T)$ with $H(T)$. The authors of [AFDM17] proposed to use a variational inference to construct a lower bound of Eq. (13). Note that these alternative solutions potentially work as loss functions but not for our explainability goals, which require a more precise estimation of compression to quantify how much information a hidden layer contains about specific features.

Therefore, a question remains: how to reliably measure the amount of input information (i.e., leakages) compressed by hidden layers when $\mathcal{X}$ and T are high dimensional data? To analyze deep neural network dynamics in profiling SCA, we look into alternative representations of input information (other than $\mathcal{X}$). When the dataset is protected with a Boolean masking countermeasure, the calculation of $I(\mathcal{X};T)$ should also indicate how much information an intermediate representation T still contains from the input secret shares existing in input traces $\mathcal{X}$ after compressing noise. As we cannot provide a correct estimation of mutual information $I(\mathcal{X};T)$ due to the reasons mentioned above, we may simplify the calculation of $I(\mathcal{X};T)$ by estimating only the amount of information that an intermediate representation T contains about a certain secret share, which is given by a feature or label set $\mathcal{Y}_m$. Thus, we replace the whole input data $\mathcal{X}$ by $\mathcal{Y}_m$ in mutual information calculation between input data and intermediate representation. Note, however, that the trained profiling model is never supplied with labels $\mathcal{Y}_m$ during the profiling or attack phases. In other words, profiling set $\mathcal{X}_p$ and attack set $\mathcal{X}_a$ are labeled from a black-box leakage model where no masks are taken into account to define $\mathcal{Y}$. The secret share labels $\mathcal{Y}_m$ (that can be a set of mask shares $\mathtt{m}$ or a set of masked S-box outputs $\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}$) are only considered for explainability purposes without affecting the profiling phase.

The information that an intermediate representation T of a given hidden layer has about secret shares $\mathcal{Y}_m$ becomes:

$$I(T;\mathcal{Y}_m) = H(\mathcal{Y}_m) - H(\mathcal{Y}_m|T), \tag{16}$$

where T can still be seen as an encoded version of $\mathcal{X}$ as the model is trained with $\mathcal{X}$. For a profiling set $\mathcal{X}_p$, the output of a hidden layer l provides an encoded version of $\mathcal{X}_p$, which is X_p^l. This way, the original profiling (resp. attack) set, which has original dimension $n_p \times J$ (resp. $n_a \times J$) where J is the number of samples in $\mathtt{x}_i$, is converted into an encoded profiling set X_p^l (resp. X_a^l) from layer l, with dimension $n_p \times w$ (resp. $n_a \times w$), where w is the number of units in the layer (for dense layers, w is given by the number of neurons while in convolution or pooling layers w is given by the number of units in a feature map). Figure 3 illustrates an example of the trace encoding process for a 3-layer MLP where all hidden layers contain 200 neurons and, therefore, $w = 200$.

Let us assume that the intermediate representation T is now X_p^l, and the amount of information that we obtain about $\mathcal{Y}_m$ when observing X_p^l is given by:

$$I(X_p^l;\mathcal{Y}_m) = H(\mathcal{Y}_m) - H(\mathcal{Y}_m|X_p^l). \tag{17}$$

The conditional entropy $H(\mathcal{Y}_m|X_p^l)$ is given by:

$$H(X_p^l|\mathcal{Y}_m) = \sum_{j=1}^{|\mathcal{Y}_m|} \mathtt{p}(\mathtt{y}_m = j) \sum_{i=1}^{n_p} \mathtt{p}(\mathtt{x}_i^l|\mathtt{y}_m = j) \log_2 \mathtt{p}(\mathtt{y}_m = j|\mathtt{x}_i^l), \tag{18}$$

where $\mathtt{x}_i^l$ is the i-th observation of X_p^l and $\mathtt{y}_m$ is one element of $\mathcal{Y}_m$.

In our method, we are interested in estimating the amount of information from secret shares that can be extracted by each hidden layer. Thus, by taking the encoded representations X_p^l from all hidden layers, at the end of each training epoch with a deep neural network model M, we train a separate classifier M_s^l with X_p^l. The classifier M_s^l can be any supervised classification method. In our case, we use a simple classifier that has an input layer connected to a Softmax layer, and it is trained for ten epochs, which was empirically verified to be sufficient to obtain satisfactory results. This process is repeated three times, and each time X_p^l is labeled differently. First and second, X_p^l is labeled with $\mathcal{Y}_{m1}$ and $\mathcal{Y}_{m2}$ (for the two secret shares in a first-order Boolean masking scheme), respectively. Third, the encoded profiling set is labeled with black-box labels $\mathcal{Y}$

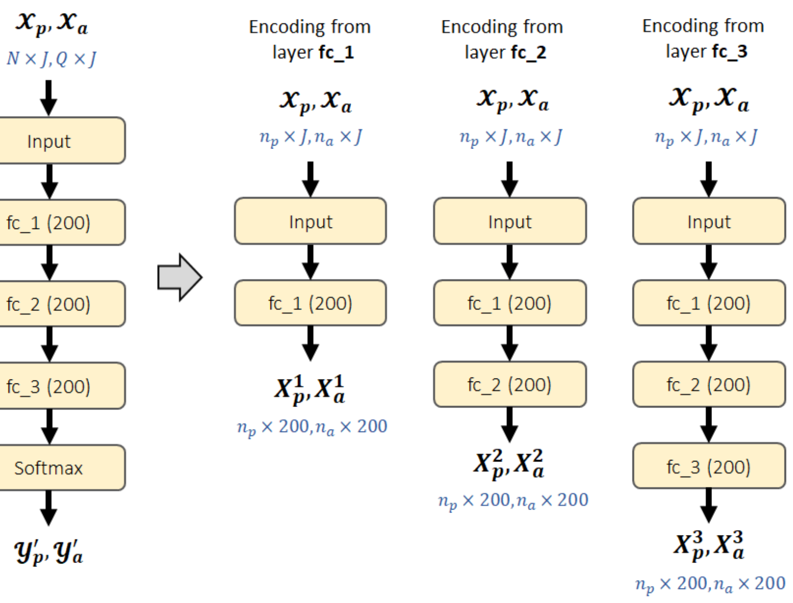

$\mathcal{X}_p, \mathcal{X}_a$
$N \times J, Q \times J$
Input
fc_1 (200)
fc_2 (200)
fc_3 (200)
Softmax
$\mathcal{Y}'_p, \mathcal{Y}'_a$
Encoding from layer **fc_1**
$\mathcal{X}_p, \mathcal{X}_a$
$n_p \times J, n_a \times J$
Input
fc_1 (200)
X^1_p, X^1_a
$n_p \times 200, n_a \times 200$
Encoding from layer **fc_2**
$\mathcal{X}_p, \mathcal{X}_a$
$n_p \times J, n_a \times J$
Input
fc_1 (200)
fc_2 (200)
X^2_p, X^2_a
$n_p \times 200, n_a \times 200$
Encoding from layer **fc_3**
$\mathcal{X}_p, \mathcal{X}_a$
$n_p \times J, n_a \times J$
Input
fc_1 (200)
fc_2 (200)
fc_3 (200)
X^3_p, X^3_a
$n_p \times 200, n_a \times 200$

Figure 3: Obtaining encoded profiling traces X^l_p and attack traces X^l_a from output layer activations.

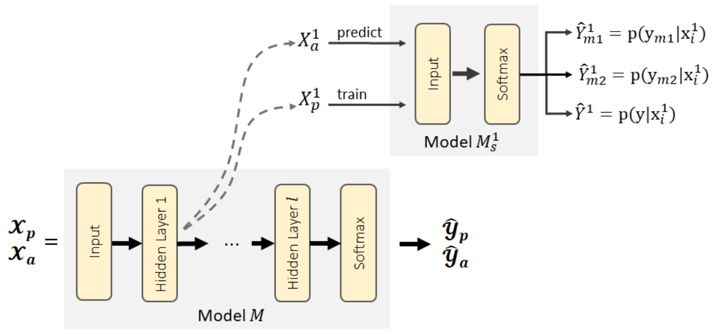

X^1_a
predict
X^1_p
train
Input
Softmax
Model M^1_s
$\hat{y}^1_{m1} = p(y_{m1}|x^1_i)$
$\hat{y}^1_{m2} = p(y_{m2}|x^1_i)$
$\hat{y}^1 = p(y|x^1_i)$
$\mathcal{X}_p$ $\mathcal{X}_a$ =
Input
Hidden Layer 1
...
Hidden Layer l
Softmax
Model M
$\hat{\mathcal{Y}}_p$ $\hat{\mathcal{Y}}_a$

Figure 4: Training and predicting classifier M^l_s after obtaining encoded representations X^l_p and X^l_a from the first hidden layer.

as we also want to understand how much information a hidden layer contains about the predictive labels. After training this simple MLP classifier M^l_s with X^l_p, we predict it with encoded attack traces X^l_a. As the process is repeated for the three different labels sets $\mathcal{Y}_{m1}$, $\mathcal{Y}_{m2}$, and $\mathcal{Y}$, the output from M^l_s provides class conditional probabilities $\mathtt{p}(\mathtt{y}_{m1}|\mathtt{x}^l_i)$, $\mathtt{p}(\mathtt{y}_{m2}|\mathtt{x}^l_i)$, and $\mathtt{p}(\mathtt{y}|\mathtt{x}^l_i)$, respectively, and $\mathtt{x}^l_i$ is now an observation of X^l_a. Due to Softmax layer in M^l_s, the class conditional probabilities are normalized as $\sum_{\mathtt{y}_{m1}} \mathtt{p}(\mathtt{y}_{m1}|\mathtt{x}^l_i) = 1$, $\sum_{\mathtt{y}_{m2}} \mathtt{p}(\mathtt{y}_{m2}|\mathtt{x}^l_i) = 1$, and $\sum_{\mathtt{y}} \mathtt{p}(\mathtt{y}|\mathtt{x}^l_i) = 1$. Note that M^l_s can also be a multi-label classifier, which provides $\mathtt{p}(\mathtt{y}_{m1}|\mathtt{x}^l_i)$, $\mathtt{p}(\mathtt{y}_{m2}|\mathtt{x}^l_i)$, and $\mathtt{p}(\mathtt{y}|\mathtt{x}^l_i)$ values with a single training of M^l_s and considerably speeds up the whole process. Figure 4 illustrates this process with an example for a single hidden layer.

After obtaining the conditional class probabilities $\mathtt{p}(\mathtt{y}_{m1}|\mathtt{x}^l_i)$, $\mathtt{p}(\mathtt{y}_{m2}|\mathtt{x}^l_i)$, and $\mathtt{p}(\mathtt{y}|\mathtt{x}^l_i)$ from M^l_s, perceived information becomes a convenient approach as it measures the amount of information learned by a profiling model with respect to specific leakage model resulting in a set of labels. The profiling model, in this case, is given by the conditional class probabilities obtained from M^l_s. First, we replace X^l_p by attack (or test) encoded representations X^l_a in

Eq. (18):

$$
\begin{aligned}
H(X_a^l|\mathcal{Y}_{m1}) &= \sum_{j=1}^{|\mathcal{Y}_{m1}|} \mathtt{p}(\mathtt{y}_{m1}=j) \sum_{i=1}^{n_a} \mathtt{p}(\mathtt{x}_i^l|\mathtt{y}_{m1}=j) \log_2 \mathtt{p}(\mathtt{y}_{m1}=j|\mathtt{x}_i^l) \\
H(X_a^l|\mathcal{Y}_{m2}) &= \sum_{j=1}^{|\mathcal{Y}_{m2}|} \mathtt{p}(\mathtt{y}_{m2}=j) \sum_{i=1}^{n_a} \mathtt{p}(\mathtt{x}_i^l|\mathtt{y}_{m2}=j) \log_2 \mathtt{p}(\mathtt{y}_{m2}=j|\mathtt{x}_i^l) \qquad (19) \\
H(X_a^l|\mathcal{Y}_m) &= \sum_{j=1}^{|\mathcal{Y}|} \mathtt{p}(\mathtt{y}=j) \sum_{i=1}^{n_a} \mathtt{p}(\mathtt{x}_i^l|\mathtt{y}=j) \log_2 \mathtt{p}(\mathtt{y}=j|\mathtt{x}_i^l),
\end{aligned}
$$

where $\mathtt{y}_{m1}$, $\mathtt{y}_{m2}$, and $\mathtt{y}$ are labels obtained from intermediate values processed in attack traces $\mathcal{X}_a$. Following the approach proposed in [BHM+19], we can replace the true probability mass function (PMF), $\mathtt{p}(\mathtt{y}_m|\mathtt{x}_i^l)$, by $\hat{\mathtt{p}}(\mathtt{y}_m|\mathtt{x}_i^{l,\mathtt{y}_m})$ and compute the perceived information by *sampling* according to (see Eq. (11) from [BHM+19]):

$$
\begin{aligned}
\widehat{PI}(X_a^l|\mathcal{Y}_{m1}) &= H(\mathcal{Y}_{m1}) + \sum_{j=1}^{|\mathcal{Y}_{m1}|} \mathtt{p}(\mathtt{y}_{m1}=j) \frac{1}{n_{\mathtt{y}_{m1}=j}} \sum_{i=1}^{n_a} \log_2 \hat{\mathtt{p}}(\mathtt{y}_{m1}|\mathtt{x}_i^{l,\mathtt{y}_{m1}=j}) \\
\widehat{PI}(X_a^l|\mathcal{Y}_{m2}) &= H(\mathcal{Y}_{m2}) + \sum_{j=1}^{|\mathcal{Y}_{m2}|} \mathtt{p}(\mathtt{y}_{m2}=j) \frac{1}{n_{\mathtt{y}_{m2}=j}} \sum_{i=1}^{n_a} \log_2 \hat{\mathtt{p}}(\mathtt{y}_{m2}|\mathtt{x}_i^{l,\mathtt{y}_{m2}=j}) \qquad (20) \\
\widehat{PI}(X_a^l|\mathcal{Y}) &= H(\mathcal{Y}) + \sum_{j=1}^{|\mathcal{Y}|} \mathtt{p}(\mathtt{y}=j) \frac{1}{n_{\mathtt{y}=j}} \sum_{i=1}^{n_a} \log_2 \hat{\mathtt{p}}(\mathtt{y}|\mathtt{x}_i^{l,\mathtt{y}=j}),
\end{aligned}
$$

where $\hat{\mathtt{p}}(y_{m1}|x_a^{l,\mathtt{y}_{m1}})$ (resp. $\hat{\mathtt{p}}(y_{m2}|x_a^{l,\mathtt{y}_{m2}})$ and $\hat{\mathtt{p}}(y|x_a^{l,\mathtt{y}})$) gives the probability that an encoded attack trace $\mathtt{x}_i^{l,\mathtt{y}_{m1}=j}$ (resp. $\mathtt{x}_i^{l,\mathtt{y}_{m2}=j}$ and $\mathtt{x}_i^{l,\mathtt{y}=j}$) is labeled with class $\mathtt{y}_{m1}$ (resp. $\mathtt{y}_{m2}$ and $\mathtt{y}$) when it is actually labeled with this same class.[6] The term $n_{\mathtt{y}_{m1}=j}$ (resp. $n_{\mathtt{y}_{m2}=j}$ and $n_{\mathtt{y}=j}$) gives the number of attack traces that are labeled with class $\mathtt{y}_{m1}=j$ (resp. $\mathtt{y}_{m2}=j$ and $\mathtt{y}=j$).

The metrics from Eq. (20) will, in the end, become an indirect estimation of how much information the encoded hidden layer representation X_p^l (obtained by predicting the neural network with profiling set $\mathcal{X}_p$) contains from secret shares labels $\mathcal{Y}_{m1}$ and $\mathcal{Y}_{m2}$ when attacking an implementation protected with a first-order masking scheme.

Obviously, the quantities $\widehat{PI}(X_a^l|\mathcal{Y}_{m1})$, $\widehat{PI}(X_a^l|\mathcal{Y}_{m2})$, and $\widehat{PI}(X_a^l|\mathcal{Y})$ most precisely indicate the quality of model M_s^l. In our case, we consider an MLP with a single hidden layer to implement M_s^l. Nevertheless, the higher the quality of M_s^l, the more precise the estimation of perceived information quantities from Eq. (20).

4.2 The Explainability Methodology

After explaining how we estimate perceived information between labels ($\mathcal{Y}_{m1}$, $\mathcal{Y}_{m2}$, and $\mathcal{Y}$) and intermediate network representations X_a^l from a hidden layer l, we can define the complete structure of our explainability methodology. Figure 5 provides the four main steps that form our methodology:

1. In Step $\textcircled{1}$, we define a baseline neural network M to be trained for E epochs with profiling traces $\mathcal{X}_p$ that are labeled as $\mathcal{Y}$ (without the knowledge of secret mask shares). The results in Sections 5 and 6 are obtained from software AES128 implementations and, therefore, (black-box) labels $\mathcal{Y}$ are generated from `S-box` output bytes in the first encryption round, i.e., $\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i)$.

[6]Here, we assume a known-key attack setting analysis. For an evaluator, the attack set is always the validation set, and the corresponding validation key bytes are always known.

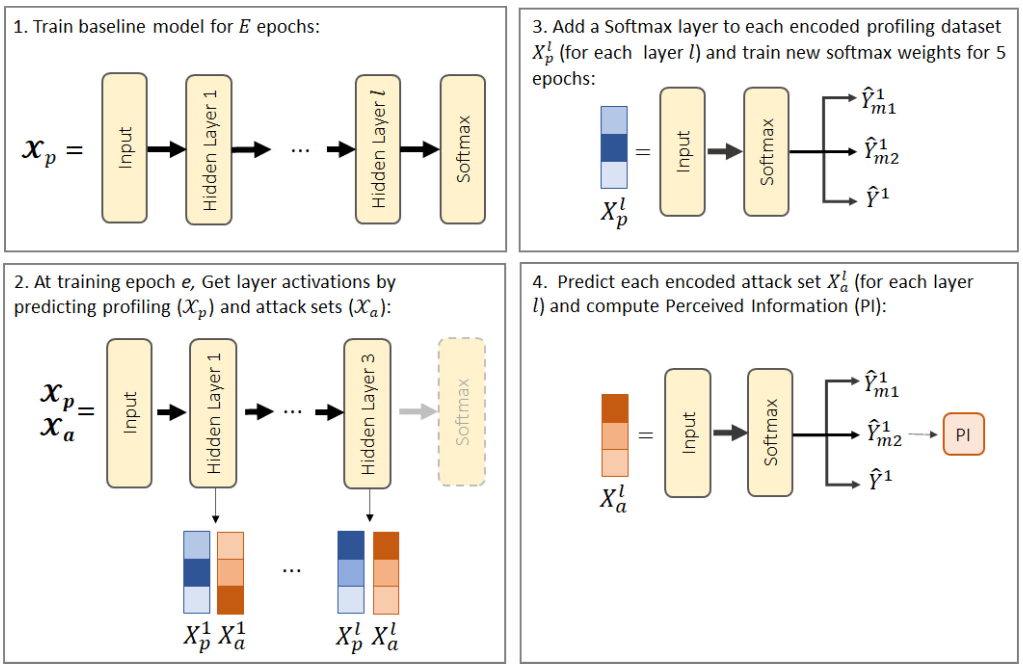

1. Train baseline model for E epochs:
$\mathcal{X}_p =$
Input
Hidden Layer 1
...
Hidden Layer l
Softmax
3. Add a Softmax layer to each encoded profiling dataset X_p^l (for each layer l) and train new softmax weights for 5 epochs:
X_p^l
=
Input
Softmax
$\hat{Y}^1_{m1}$
$\hat{Y}^1_{m2}$
$\hat{Y}^1$
2. At training epoch e, Get layer activations by predicting profiling ($\mathcal{X}_p$) and attack sets ($\mathcal{X}_a$):
$\mathcal{X}_p =$
$\mathcal{X}_a$
Input
Hidden Layer 1
...
Hidden Layer 3
Softmax
...
X_p^1 X_a^1
X_p^l X_a^l
4. Predict each encoded attack set X_a^l (for each layer l) and compute Perceived Information (PI):
X_a^l
=
Input
Softmax
$\hat{Y}^1_{m1}$
$\hat{Y}^1_{m2}$
$\hat{Y}^1$
PI

Figure 5: Methodology for mask share fitting explainability.

2. In Step $\textcircled{2}$, we extract the intermediate representations from hidden layers. Thus, at the end of training epoch e, the trained model M predicts both profiling ($\mathcal{X}_p$) and attack ($\mathcal{X}_a$) sets. For each hidden layer l, we obtain output activations that are taken as encoded versions of input profiling and attack sets, i.e., X_p^l and X_a^l, respectively.
3. In Step $\textcircled{3}$, we take encoded profiling sets X_p^l from all hidden layers l and build a simple classifier, denoted as M_s^l, which is trained with X_p^l for ten epochs. This classifier consists of an input connected to a Softmax classifier as illustrated in Figure 4. We consider only ten epochs as it was empirically verified to be sufficient to obtain satisfactory results. The training of the M_s^l classifier is relatively fast. There are no pre-processing steps such as dimensionality reduction, which could be computation-intensive in the case a large output dimension from a hidden layer produces a very large encoded set X_p^l. Therefore, using machine learning algorithms (e.g., Support Vector Machine or Decision Trees) or even Gaussian Template Attacks to implement the M_s^l classifier would require feature selection and/or dimensionality reduction, which could render the process impractical due to significant overheads. As such, M_s^l classifier is trained three times, each time X_p^l is labeled with a different label set, i.e., $\mathcal{Y}_{m1}$, $\mathcal{Y}_{m2}$, or $\mathcal{Y}$. As already mentioned in the previous section, instead of training M_s^l three times, one could simply implement a multi-label MLP classifier to speed up the process.
4. Finally, in Step $\textcircled{4}$, each classifier M_s^l predicts encoded attack set X_a^l, producing output class probabilities $\widehat{\mathcal{Y}^l_{m1}} = \mathtt{p}(\mathtt{y}_{m1}|\mathtt{x}_i^l)$, $\widehat{\mathcal{Y}^l_{m2}} = \mathtt{p}(\mathtt{y}_{m2}|\mathtt{x}_i^l)$, and $\widehat{\mathcal{Y}^l} = \mathtt{p}(\mathtt{y}|\mathtt{x}_i^l)$. These quantities are considered for a perceived information calculation in Eq. (20).

This way, we can estimate the amount of information that an encoded representation X_p^l, in a hidden layer l, can have about input information $\mathcal{Y}_{m1}$, $\mathcal{Y}_{m2}$, or $\mathcal{Y}$ for every training epoch in a black-box model. Algorithm 1 provides the steps necessary to implement our explainability methodology. The method `LayerPredict`$(L_m, \mathcal{X})$ returns the output activations from a layer L_m of index l when this layer predicts some input $\mathcal{X}$.

Algorithm 1 Steps for mask share fitting explainability.

```
1: procedure 2-Share Mask Explainability(model $M$, Softmax model $M_s$, number of layers $L$ in model $M$, number of epochs $E_M$ for model $M$, number of epochs $E_S$ for model $M_s^l$, profiling set $\mathcal{X}_p$, attack set $\mathcal{X}_a$, mask labels $\mathcal{Y}_{m1}$, mask labels $\mathcal{Y}_{m2}$, labels $\mathcal{Y}$)
2:     for $e_m = 1$ to $E$ do
3:         $M_{e_m} \leftarrow$ TrainOneEpoch($M$, $\mathcal{X}_p$)                                ▷ Step 1 in Figure 5
4:         for $l = 1$ to $L$ do
5:             $L_M \leftarrow$ GetLayer($M_{e_m}$, $l$)                                   ▷ Step 2 in Figure 5
6:             $X_p^l \leftarrow$ LayerPredict($L_M$, $\mathcal{X}_p$)                          ▷ Step 2 in Figure 5
7:             $X_a^l \leftarrow$ LayerPredict($L_M$, $\mathcal{X}_a$)                          ▷ Step 2 in Figure 5
8:             $M_s^l \leftarrow$ Train($M_s$, $E_S$, $X_p^l$, $\mathcal{Y}_{m1}$)                     ▷ Step 3 in Figure 5
9:             $\hat{\mathcal{Y}}_{m1}^l \leftarrow$ Predict($M_s^l$, $X_a^l$)                            ▷ Step 4 in Figure 5
10:            $\hat{\mathcal{Y}}_{m2}^l \leftarrow$ Predict($M_s^l$, $X_a^l$)                            ▷ Step 4 in Figure 5
11:            $\hat{\mathcal{Y}}^l \leftarrow$ Predict($M_s^l$, $X_a^l$)                              ▷ Step 4 in Figure 5
12:            $\widehat{PI}(X_a^l; \hat{\mathcal{Y}}_{m1}^l)$ = PerceivedInformation($X_a^l$, $\hat{\mathcal{Y}}_{m1}^l$)   ▷ Step 4 in Figure 5
13:            $\widehat{PI}(X_a; \hat{\mathcal{Y}}_{m2}^l)$ = PerceivedInformation($X_a^l$, $\hat{\mathcal{Y}}_{m2}^l$)   ▷ Step 4 in Figure 5
14:            $\widehat{PI}(X_a; \hat{\mathcal{Y}}^l)$ = PerceivedInformation($X_a^l$, $\hat{\mathcal{Y}}^l$)       ▷ Step 4 in Figure 5
15:        end for
16:    end for
17: end procedure
```

4.3 On the Granularity of the Explainability Approach

One could ask why developing an explainability approach that works on the granularity level of a layer. For instance, previous approaches allowed consideration even for specific neurons (filters in a feature map). Our analysis showed that multiple neurons (filters in a feature map) are responsible for the successful deep learning-based SCA. Thus, trying to connect specific neurons (filters in a feature map) to the outcome of the analysis is difficult due to the ability of neural networks to find (many) good functions representing the leakage.
Contrary, looking at the level of multiple layers allows the analysis but does not provide information on what happens in every layer, which is especially relevant if we consider that state-of-the-art neural networks in SCA contain only a few layers [PWP21].

5 Compression in Deep Neural Networks

As explained in Section 2.3.2, the information bottleneck theory suggests that one of the main aspects of learning is compression of $\mathcal{X}$ during training. The measure of compression in deep neural networks is given by the reduction of mutual information $I(\mathcal{X};T)$ as the training process evolves. At the beginning of training, $I(\mathcal{X};T)$ is maximal for a given representation T (i.e., the output of a hidden layer), which has all possible information about $\mathcal{X}$. As soon as the model starts to learn, T becomes a bottleneck and starts to lose information about $\mathcal{X}$ (ideally, noise) by preserving only the relevant features which are necessary to predict $\mathcal{Y}$. As detailed in the previous section, a precise measure of compression by computing $I(\mathcal{X};T)$ is particularly hard due to the dimensionality of $\mathcal{X}$ and T, especially for histogram-based mutual information approaches.

As we are interested in understanding the processing of secret shares by hidden layers during training, instead of measuring the compression of the whole profiling set $\mathcal{X}_p$ by a hidden layer, we measure the compression of specific information contained in $\mathcal{X}_p$, which are the features related to different intermediate values, including the secret shares $\mathcal{Y}_m$. For that, we adopted perceived information which is a lower bound of mutual information. The problem that arose is that we need to obtain the conditional probabilities $\texttt{p}(\texttt{y}_m|\texttt{x})$ from an intermediate network representation X_p^l, which is the output of a layer l. As specified in line 8 in Algorithm 1, obtaining $\texttt{p}(\texttt{y}_m|\texttt{x})$ requires the training of a separate M_s^l classifier with X_p^l and predicted with X_a^l. Thus, the M_s^l classifier can estimate the amount of information that X_p^l (an intermediate network representation) has about a secret share $\mathcal{Y}_m$, which is a feature present in the profiling set $\mathcal{X}_p$. Intuitively, one would expect to maximize $\widehat{PI}(X_a^l;\mathcal{Y}_m)$ for all secret shares in all the hidden layers when $\mathcal{Y}_m$ is a feature to be learned to predict $\mathcal{Y}$. This would suggest that compression of $I(\mathcal{X};T)$ is happening because a hidden layer is increasing the information about a secret share $\mathcal{Y}_m$. This is more likely to happen when the intermediate representation X_p^l compresses noise or irrelevant features. Moreover, in our results, we will differentiate compression from overfitting. Overfitting should be characterized by a decrease of $\widehat{PI}(X_a^l;\mathcal{Y}_m)$ for every feature $\mathcal{Y}_m$, including those representing the secret shares that have to be learned to predict $\mathcal{Y}$. Otherwise, when $\widehat{PI}(X_a^l;\mathcal{Y}_m)$ decreases only for features $\mathcal{Y}_m$ that are not related to $\mathcal{Y}$, we assume that compression is happening.

5.1 Compression of Irrelevant (Key Byte) Features in First-order Masked Datasets

Next, we use our explainability methodology to show how deep neural network layers compress the information about irrelevant features present in $\mathcal{X}_p$ during training while preserving relevant ones. When targeting a single key byte in a first-order masked dataset, relevant features become the two secret shares associated with the target key byte. In contrast, irrelevant features are all the information corresponding to other key bytes and noise components. We provide results for a noise-free simulated dataset, in which all features are well defined so that every sample represents a feature and there are no samples that would represent noise. Section 6 provides experiments on real side-channel measurements from first-order Boolean masked AES implementations.

Our simulated traces contain leakages from `S-box` outputs in the first AES encryption round. Each trace $\texttt{x}_i$ contains 32 samples, and each of these samples is generated according to the following equations:

$$\begin{aligned} \texttt{x}_i[2j] &= \texttt{S-box}(\texttt{d}_j \oplus \texttt{k}_j) \oplus \texttt{m}_j \\ \texttt{x}_i[2j+1] &= \texttt{m}_j, \end{aligned} \tag{21}$$

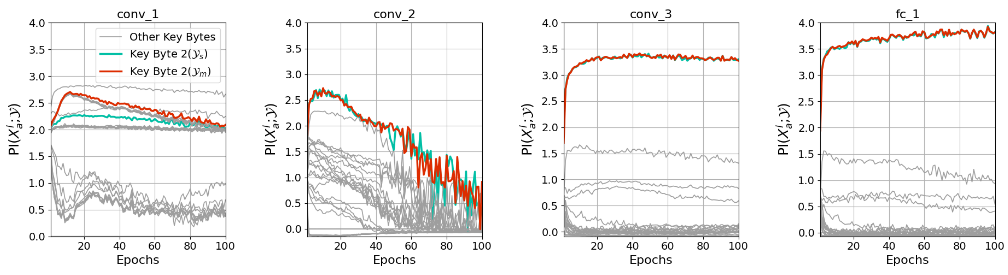
where $j \in [0, 15]$ denotes the j-th key byte index and $\texttt{m}_j$ is the mask share associated with the j-th key byte. In this case, every trace sample represents a feature, which will be denoted as $\mathcal{Y}_m[j]$ and $\mathcal{Y}_s[j]$ for the mask share $\texttt{m}_j$ and $\texttt{S-box}(\texttt{d}_j \oplus \texttt{k}_j) \oplus \texttt{m}_j$, respectively.

In the first experiment, we ran a random hyperparameter search (see Appendix A, Table 2) with the simulated dataset and found the CNN with the following layer-wise structure:

$$\mathcal{X} \rightarrow [C(\texttt{fi}, 10, 3) \rightarrow E \rightarrow BN \rightarrow AP(2,2)]^3 \rightarrow [FC(100) \rightarrow E]^1 \rightarrow S(256) \rightarrow \widehat{\mathcal{Y}},$$

where `fi` is the number of filters in each convolution layer, which is set to 16, 32, and 48. Learning rate and batch size are arbitrarily set to 0.005 and 400, respectively. This CNN is trained with `Adam` optimizer for 100 epochs. The three convolution layers are named `conv_1`, `conv_2`, and `conv_3` and the fully-connected layer is named `fc_1`. The simulated profiling and attack traces $\mathcal{X}_p$ and $\mathcal{X}_a$ are labeled with leakages from the second key byte $j = 2$, i.e., $\mathcal{Y} = \texttt{S-box}(\texttt{d}_2 \oplus \texttt{k}_2))$.

In Figure 6, we plot the $\widehat{PI}(X_a^l;\mathcal{Y}_s[j])$ and $\widehat{PI}(X_a^l;\mathcal{Y}_m[j]))$ values obtained for all convolution (`conv_1`, `conv_2`, and `conv_3`) and fully-connected (`fc_1`) layers. The perceived



(a) `conv_1` **(b)** `conv_2` **(c)** `conv_3` **(d)** `fc_1`

Figure 6: The compression of irrelevant features with CNN. Red and green lines indicate the perceived information from relevant features, which are the mask $\mathcal{Y}_m[2]$ and the masked `S-Box` output $\mathcal{Y}_s[2]$ labels related to key byte 2. Irrelevant features associated with the rest of the key bytes are illustrated as gray lines.

information values are computed with respect to all features $\mathcal{Y}_m[j]$ and $\mathcal{Y}_s[j]$ (for $j \in [0, 15]$) contained in simulated traces. Because the model is trained to predict $\mathcal{Y} = \texttt{S-box}(\texttt{d}_2 \oplus \texttt{k}_2)$), we expect that the network layers will fit the relevant features corresponding to key byte 2, i.e., $\mathcal{Y}_m[2]$ and $\mathcal{Y}_s[2]$, and discard the rest of the existing and irrelevant features. When we analyze Figure 6a, we can see that convolution layer `conv_1` still contains information from several irrelevant features from other key bytes (other than key byte 2). When we move our analysis to the convolution layer `conv_2` in Figure 6b, we recognize a bottleneck being implemented by the CNN, in which irrelevant features start to be compressed in this layer. In contrast, the relevant ones are kept with relatively more information. Then, when we look at `conv_3` in Figure 6c, we clearly see a bottleneck layer compressing most of irrelevant features (although for some key bytes some of these features still show positive perceived information) while trying to maximize the relevant ones (which are $\widehat{PI}(X_a^l; \mathcal{Y}_s[2])$ and $\widehat{PI}(X_a^l; \mathcal{Y}_m[2])$)) to improve prediction of $\mathcal{Y}$. Finally, in the fully-connected layer `fc_1`, the network already has information from relevant features to provide a good generalization to $\mathcal{Y}$. Therefore, this is a clear example of an information bottleneck being implemented by a deep neural network, and applying our explainability methodology allows us to explain what happens in this model during training.

6 Experimental Results

In this section, we provide experimental results on a variety of CNN and MLP configurations for `ASCADr` and `DPAv4.2` datasets. For both datasets, we target trace intervals with second-order leakages of multiple key bytes to measure the compression of irrelevant features during training in different hidden layers.

6.1 Reading the Plots

Here, the main goal is to demonstrate how different hidden layers fit, compress, or generalize with respect to different features. Plots provided in this section show the evolution of perceived information (Eq. (20)) during training for different features given by specific label sets. For both evaluated datasets (`ASCADr` and `DPAv4.2`), we deploy profiling and attack phases over trace intervals that include leakages from different key bytes. For `ASCADr`, the evaluated interval includes second-order leakages from key bytes 2, 4, 5, and 11, in which the target is key byte 2. For `DPAv4.2`, the target interval includes second-order leakages from key bytes 0, 4, 5, 9, and 12 and we target key byte 0. The main idea is to illustrate the compression of irrelevant features by bottleneck layers. Thus, in all plots, we provide perceived information results for masks shares (given by $\mathcal{Y}_m[j]$ in the plot's legend, where j is the key byte index) and masked `S-Box` output byte (given by $\mathcal{Y}_s[j]$ in plot's legend) for all these key bytes. The perceived information values for target key bytes (key byte 2 for `ASCADr` and key byte 0 for `DPAv4.2`) are shown with colored lines (red color for $\mathcal{Y}_m[j]$ and green color for $\mathcal{Y}_s[j]$) while for the rest of key bytes we show the results with gray lines. The perceived information for the actual black-box attack labels $\mathcal{Y}$ is represented by an orange line plot. Therefore, colored lines indicate relevant features while gray lines indicate irrelevant features. Every subfigure shows results for a specific hidden layer.

6.2 `ASCADr`

6.2.1 Multilayer Perceptron

We select various MLP architectures that implement successful key recovery on `ASCADr` from a random search (see Appendix 1 for details). We consider a profiling MLP model to be successful when it reaches guessing entropy equal to 0 for the correct key $\mathtt{k}_2$ after processing up to 5 000 attack traces. Our hyperparameter search process allows us to find successful models with a different number of hidden layers. Nevertheless, by applying our explainability methodology, we observe common behavior for these models regardless of the number of hidden layers.

Figure 7 shows an example for a six-layer MLP with the following layer-wise structure:

$$\mathcal{X} \rightarrow [FC(\mathtt{100}) \rightarrow E]^6 \rightarrow S(\mathtt{256}) \rightarrow \widehat{\mathcal{Y}}. \tag{22}$$

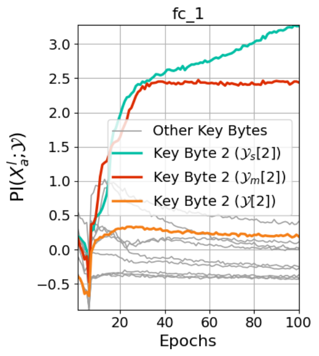
For this model, learning rate is set to `5e-4` and weights are initialized with `random_uniform` method. The attacked interval includes the second-order leakages from four different key bytes, including the target one. We immediately verify that the first hidden layer still contains information from irrelevant features represented by key bytes different from the target one. From the second layer, we see that the irrelevant information is highly compressed, and the outer layer generalizes better to $\mathcal{Y}$, as shown by the orange line representing $\widehat{PI}(X_a^l;\mathcal{Y})$. Another interesting fact from this specific model is that outer layers achieve higher values of $\widehat{PI}(X_a^l;\mathcal{Y})$ even before achieving higher values for $\widehat{PI}(X_a^l;\mathcal{Y}_s[2])$ and $\widehat{PI}(X_a^l;\mathcal{Y}_m[2])$. This happens because previous layers already implemented unmasking and transmitted this information to the next layer.

Answering the **ExDL-SCA** questions, we provide the following observations:

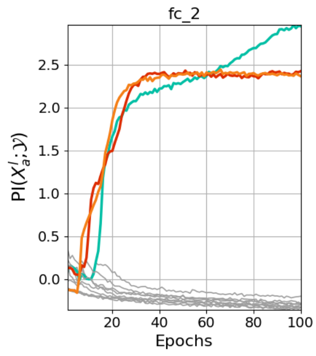
1. **Where**. Our empirical results indicate that compression of $\mathcal{X}$ mostly happens in the first hidden layer in any MLP configuration. Generalization to $\mathcal{Y}$ is stronger in hidden layers closer to the output layer, and this conclusion comes from higher $\widehat{PI}(X_a^l;\mathcal{Y})$ values obtained for the outer layer in comparison to hidden layers closer to the input layer.
2. **What**. We verified that to generalize to $\mathcal{Y}$, the first hidden layer compresses noise and irrelevant features and transmits information from relevant secret shares to the subsequent hidden layers. This also suggests that hidden layers perform unmasking by combining the two secret shares.
3. **Why**. Our analyses indicate that MLP implements IB theory as the bottleneck is usually implemented already by the first hidden layer. In essence, all intermediate network representations follow the IB theory, but the bottleneck (compression of irrelevant features) is usually more evident in the first layer.

6.2.2 Convolutional Neural Network

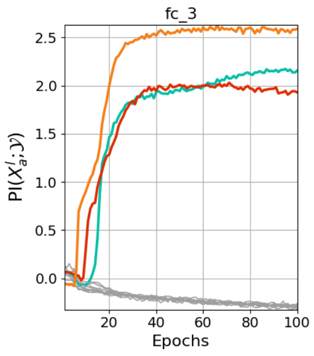
We again deployed a random search to select various CNN architectures that implement successful key recovery on the `ASCADr` dataset. Details about our CNN random search



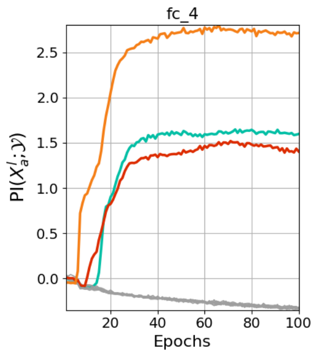
(a) `fc_1`



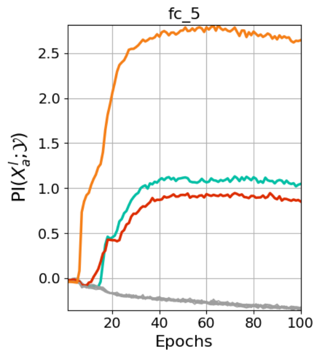
(b) `fc_1`



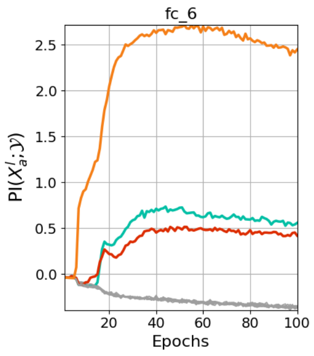
(c) `fc_1`



(d) `fc_1`



(e) `fc_1`



(f) `fc_1`

Figure 7: Perceived information values from a six-layer MLP trained with the `ASCADr` dataset.

process can be found in Appendix A (Table 2). The number of convolution layers in the search space ranges from one to four, and CNN models may contain one or two fully-connected layers.

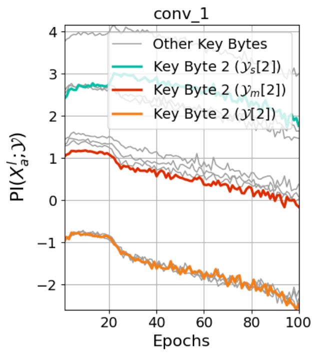
Figure 8 shows the results obtained from a CNN with four convolution layers and two fully-connected layers with the following structure:

$$\mathcal{X} \rightarrow [C(\mathtt{fi}, 40, 15) \rightarrow SE \rightarrow BN \rightarrow AP(2,2)]^4 \rightarrow [FC(20) \rightarrow SE]^2 \rightarrow S(256) \rightarrow \widehat{\mathcal{Y}}, \qquad (23)$$

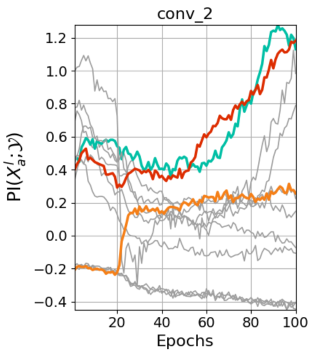
where `fi` is set to 12, 24, 36, and 48 for the four convolution layers. The learning rate for this model is `1e-4` and trainable weights are initialized with `glorot_normal` method. The first layer, `conv_1`, fits information from relevant and irrelevant features, as perceived information values are positive. For this layer, after epoch 20, compression starts to happen for all features. Layer `conv_2` shows the fitting of input features, including irrelevant ones, and layer `conv_3` shows compression of irrelevant features while preserving and learning relevant ones. Note how `conv_3` already generalized to Y. The subsequent layers (`conv_4`, `fc_1`, and `fc_2`) also show compression of irrelevant leakages from key bytes other than the target one. Prediction to $\mathcal{Y}$ (given by positive values of $\widehat{PI}(X_a^l;\mathcal{Y})$), is already seen in `conv_4`, and in `fc_1` and `fc_2` layers, this generalization to $\mathcal{Y}$ is even stronger. What we see in this figure is a general behavior observed for various successful CNN models on the `ASCADr` dataset.

By applying our explainability methodology to different CNN configurations, we answer the following **ExDL-SCA** questions:

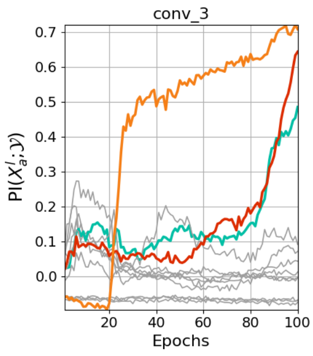
1. **Where**. By applying our explainability approach to different CNN architectures,



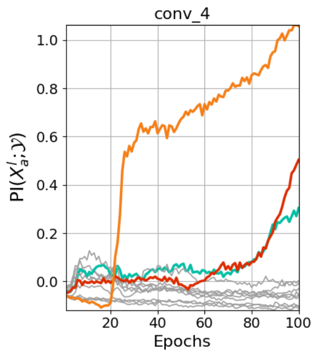
(a) `conv_1`



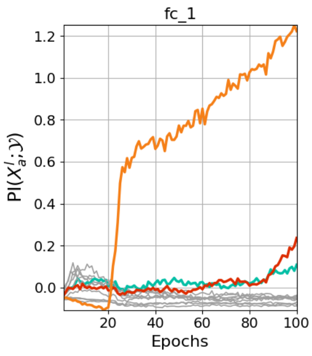
(b) `conv_2`



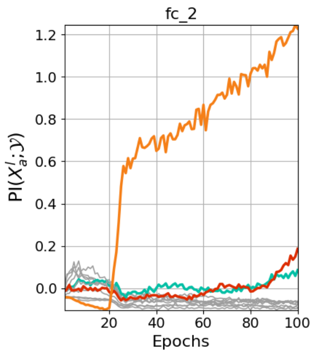
(c) `conv_3`



(d) `conv_4`



(e) `fc_1`



(f) `fc_2`

Figure 8: Perceived information values from CNN layers (27) trained with the `ASCADr` dataset. Red and green lines represent the secret shares $\mathcal{Y}_m = \mathtt{m}_2$ (the mask) and $\mathcal{Y}_s = \texttt{S-Box}(\mathtt{d}_2 \oplus \mathtt{k}_2) \oplus \mathtt{m}_2$ (the masked `S-Box` output byte), respectively. Orange line represents the black-box labels $\mathcal{Y} = \texttt{S-Box}(\mathtt{d}_2 \oplus \mathtt{k}_2)$.

we verify that the first convolution layer is usually unable to implement compression of irrelevant features to keep the relevant ones (we observed that when the model achieves good levels of generalization, the first convolution layer tends to compress the input information, including the features related to the target key byte). The second convolution layer usually implements fitting and compression phases, which is characterized by the increase of perceived information values obtained from features (i.e., secret shares) related to the target key byte. At the same time, there are reduced perceived information values for features related to the remaining key bytes that the model is not supposed to learn. When CNN has more than two convolution layers, the other convolution layers implement the bottleneck more efficiently. Fully-connected layers provide higher $\widehat{PI}(X_a^l; \mathcal{Y})$, indicating generalization to $\mathcal{Y}$.

2. **What**. The bottleneck is implemented, at least in the second hidden layer. From this moment, the network starts to generate an intermediate representation that mostly preserves the information from secret shares related to the target key byte.
3. **Why**. Fitting and compression happen in CNN models because this type of architecture also follows the IB principle. Hidden layers implement the bottleneck, which provides conditions for the model to generalize as relevant features are fit by the layers.

6.2.3 Convolutional Neural Networks with Desynchronized Dataset

In this section, we evaluate CNNs with our explainability methodology when the model is trained with a desynchronized `ASCADr` dataset. To produce misalignment, traces are randomly shifted by up to 50 samples (see [WP20] for details on how to simulate the resynchronization effect). To circumvent the desynchronization effect, the CNN is trained with data augmentation that implements random shifts (again up to 50 samples). For each epoch, we generate 200 000 augmented profiling traces. We apply our hyperparameter search until we generate at least 100 successful CNN models able to reduce the guessing entropy of the correct key to 0. In Figure 9, we provide an example result from a CNN model with the following layer-wise structure:

$$\mathcal{X} \to [C(\texttt{fi}, \texttt{30}, \texttt{15}) \to E \to BN \to AP(\texttt{2}, \texttt{2})]^3 \to [FC(\texttt{200}) \to E]^2 \to S(\texttt{256}) \to \widehat{\mathcal{Y}}, \quad (24)$$

where filters `fi` are set to 16, 32, and 48 for the three convolution layers. Learning rate is set to `1e-4` and weights are initialized with `random_uniform` method.

In Figures 9a and 9b, we see how first two convolution layers `conv_1` and `conv_2` process relevant and irrelevant features. These layers mostly fit all features, without compression. Layers `conv_3`, `fc_1`, and `fc_2` start to implement the compression of irrelevant features related to key bytes different from $\texttt{k}_2$, more specifically after epoch 50. Looking at Figures 9c, 9d, and 9e, we conclude that these three layers implement bottlenecks, but the perceived information values suggest that this model should be trained for more epochs, as we see a growing trend for relevant features and generalization (given by $\widehat{PI}(X_a^l; \mathcal{Y})$) while perceived information values related to irrelevant features are continuously decreasing.

Addressing the **ExDL-SCA** questions:

1. **where**. Bottleneck layers are usually implemented by layers closer to the output layer. When the CNN contains more than two convolution layers, we observed that the bottleneck happens from the third convolution layer.
2. **what**. The bottleneck is implemented less efficiently when the network is trained on a more noisy dataset. Irrelevant features (and probably other sources of noise) are preserved until the last hidden layer with some level of compression. Relevant features, which are necessary to defeat masking, are usually (but not always) preserved more intensively, allowing the model to implement a second-order attack successfully.
3. **why**. Our results suggest that the first convolution layers work on bypassing desynchronization effects, becoming less prone to separate relevant from irrelevant features.

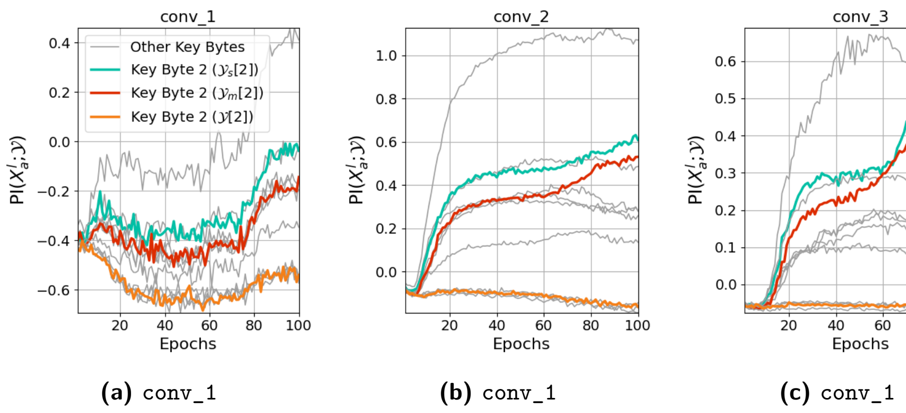
6.3 `DPAv4.2`

6.3.1 Multilayer Perceptron

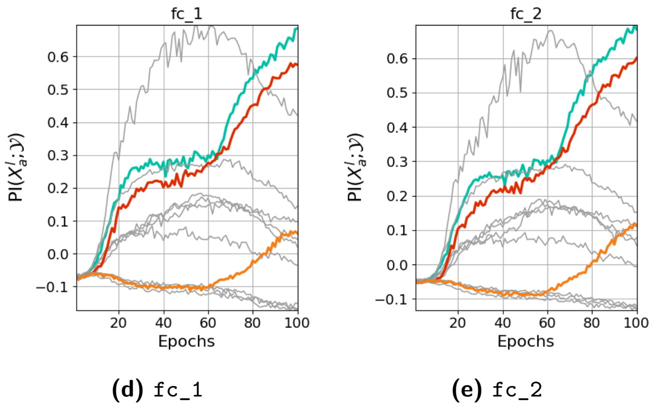
We apply the same hyperparameter search process from Appendix A (Table 1) for the `DPAv4.2` dataset to find successful MLP models. Again, for these models, we observe similar behavior when applying our explainability methodology. Figure 10 shows results for the following layer-wise MLP structure:

$$\mathcal{X} \to [FC(\texttt{20}) \to E]^3 \to S(\texttt{256}) \to \widehat{\mathcal{Y}}. \quad (25)$$

Here, the learning rate is set to `5e-4` and weights are initialized with `glorot_normal` method. We apply `l1` regularization to all hidden layers with a regularization value of $l1 = \texttt{1e-4}$. What this figure shows is a common trend observed from multiple successful MLP models we found with our hyperparameter search process. The first two hidden layers (when models contain more than two hidden layers, at least) show already the capacity to differentiate between relevant and irrelevant features, with the compression of

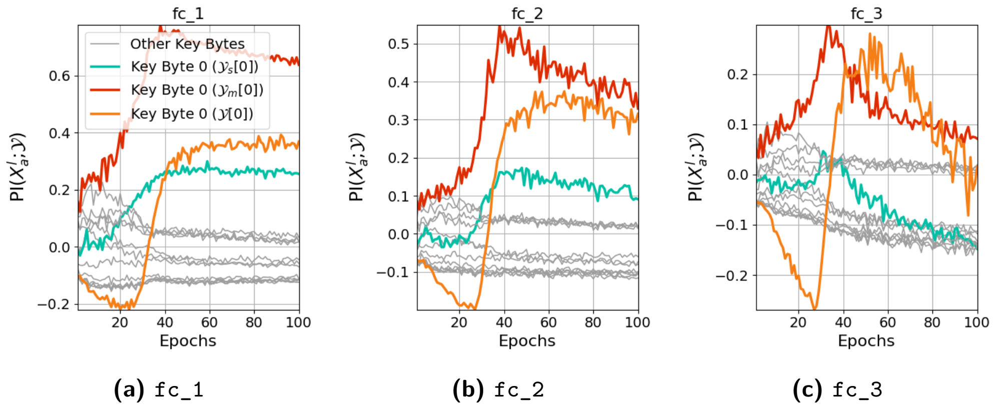


(a) `conv_1` **(b)** `conv_1` **(c)** `conv_1`



(d) `fc_1` **(e)** `fc_2`

Figure 9: Perceived information values from a CNN trained with the desynchronized `ASCADr` dataset.



(a) `fc_1` **(b)** `fc_2` **(c)** `fc_3`

Figure 10: Perceived information values from a MLP trained with the `DPAv4.2` dataset.

irrelevant ones. The perceived information values $\widehat{PI}(X_a^l; \mathcal{Y}_m[0])$ are significantly higher than $\widehat{PI}(X_a^l; \mathcal{Y}_s[0])$ in all layers, which is a consequence of higher SNR values for the mask inside the attacked interval. For the last hidden layer, `fc_3`, the values of $\widehat{PI}(X_a^l; \mathcal{Y}_s[0])$ become negative after epoch 40, which also reflects in the values of $\widehat{PI}(X_a^l; \mathcal{Y})$ that indicate prediction to $\mathcal{Y}$ and overfitting.

Addressing the **ExDL-SCA** questions, we conclude:

1. **Where**. MLP models for `DPAv4.2` implement the bottleneck already in the first fully-connected layers. As also observed for `ASCADr` dataset, the first hidden layer already implements generalization as soon as the compression phase happens.
2. **What**. The bottleneck, which is implemented already in the first hidden layers, separates relevant from irrelevant features and also shows generalization ability. Because `DPAv4.2` is a smaller dataset in terms of profiling traces, overparameterized MLP models tend to overfit the relevant features, resulting in a decrease of $\widehat{PI}(X_a^l;\mathcal{Y})$ values in outer layers.
3. **Why**. With MLPs, the first hidden layer is already able to implement the bottleneck and the generalization to $\mathcal{Y}$, suggesting that a single hidden layer would be enough to successfully implement the profiling model. Our experiments required the usage of `l1` or `l2` regularization to find better performing models, indicating that overparameterized MLPs are problematic for this dataset.

6.3.2 Convolutional Neural Network

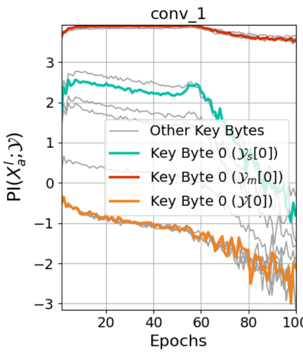
We find successful CNN models for `DPAv4.2` by applying the hyperparameter search process from Appendix A (Table 2). We identified several successful CNN models that reduce the guessing entropy of the target key byte to 0 with up to 5 000 attack traces. Figure 11 shows an example result for the CNN with the following structure:

$$\mathcal{X} \to [C(\mathtt{fi},\mathtt{40},\mathtt{10}) \to E \to BN \to AP(\mathtt{2},\mathtt{2})]^4 \to [FC(\mathtt{100}) \to E]^2 \to S(\mathtt{256}) \to \widehat{\mathcal{Y}}, \quad (26)$$

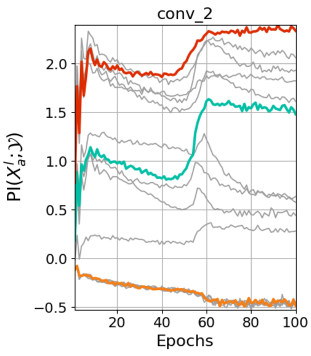
where filters `fi` are set to 4, 8, 12, and 16 for the four convolution layers. For this model, learning rate is set to `1e-3` and weights are initialized with `random_uniform` method. Similar to what we observed for `ASCADr` CNN case, for `DPAv4.2` the first convolution layers fit relevant and irrelevant features without clear compression. From convolution layer `conv_3`, we verify the occurrence of the bottleneck where relevant features are learned while irrelevant ones are compressed, as showed in Figure 11c. The subsequent layers `conv_4`, `fc_1`, and `fc_2` show significant levels of generalization to $\mathcal{Y}$, as we confirm by observing higher values of $\widehat{PI}(X_a^l;\mathcal{Y})$ (orange lines). This suggests that convolution layers also implement classification, as this task is usually attributed to fully-connected layers in the DL-SCA literature [CZLG21].

We answer the **ExDL-SCA** questions:

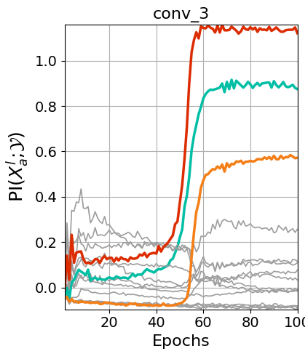
1. **Where**. We commonly observed that the first convolution layer shows the fitting of input features while compression happens for relevant and irrelevant ones. The bottleneck layer usually happens from the second or third convolution layer. Generalization to $\mathcal{Y}$ happens with more intensity in hidden layers closer to the output layer.
2. **What**. Compression of irrelevant features by a bottleneck layer tends to happen as soon as the first convolution layer starts to compress all features, including relevant ones. Note that here we are estimating compression from the full feature map obtained from each convolution layer. To verify if some of the features are not compressed in the first layer, compression should be estimated per filter in a feature map. In terms of generalization to $\mathcal{Y}$, convolution layers closer to the fully-connected layer show positive perceived information values for $\widehat{PI}(X_a^l;\mathcal{Y})$ as they received already relevant features from previous layers.
3. **Why**. As our selected CNN models can successfully implement second-order attacks, we verify that hidden layers can automatically locate relevant features from side-channel measurements by differentiating them from irrelevant ones and noise. This suggests that the information bottleneck principle is happening in these models.



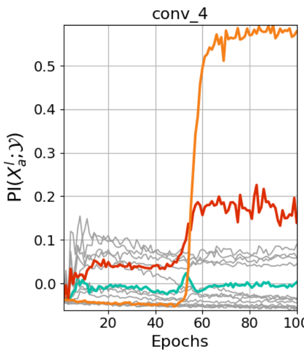
(a) `conv_1`



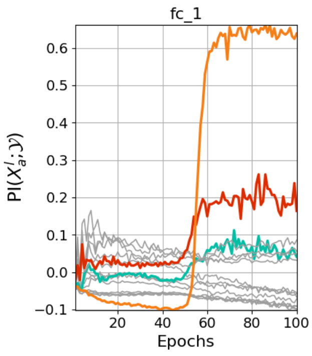
(b) `conv_2`



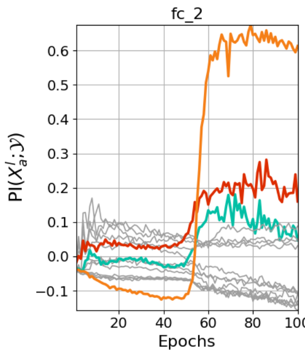
(c) `conv_3`



(d) `conv_4`



(e) `fc_1`



(f) `fc_2`

Figure 11: Perceived information values from a CNN trained with the `DPAv4.2` dataset.

6.3.3 Convolutional Neural Networks with Desynchronized Dataset

We apply again the random hyperparameter search process from Table 2 to the desynchronized `DPAv4.2` dataset. We apply random shifts to measurements with up to 50 samples and the resulting desynchronized dataset has a very low SNR. Training is performed with data augmentation, in which we generate 70 000 augmented traces per epoch. Data augmentation is based on random shifts for up to 50 samples.

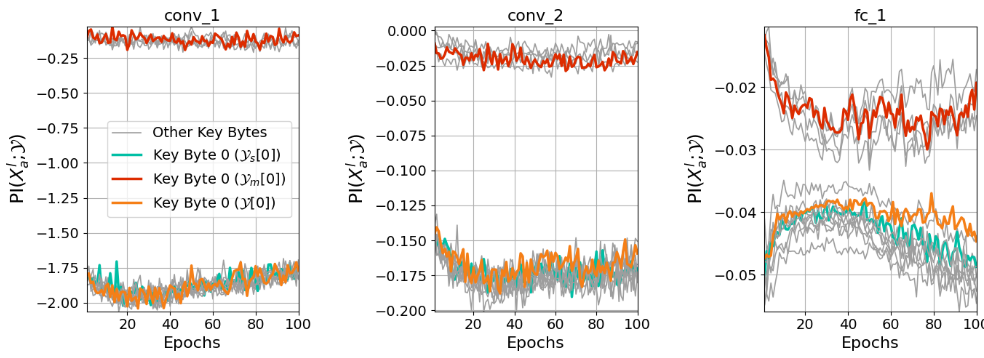
Figure 12 shows explainability results obtained from a CNN with the following layer-wise structure, which is selected among successful CNN models obtained with the random search:

$$\mathcal{X} \rightarrow [C(\mathtt{fi}, \mathtt{20}, \mathtt{15}) \rightarrow RE \rightarrow BN \rightarrow AP(\mathtt{2}, \mathtt{2})]^2 \rightarrow [FC(\mathtt{20}) \rightarrow RE]^1 \rightarrow S(\mathtt{256}) \rightarrow \widehat{\mathcal{Y}}. \tag{27}$$

Here, filters `fi` are set to 4 and 8 for the two convolution layers. Because the resulting desynchronized dataset has very low SNR, compression and fitting phases are not clearly highlighted in perceived information results. In fact, layer `fc_1` seems to implement the bottleneck as we can observe, from Figure 12c, that some of irrelevant features are compressed while still not compressing all irrelevant ones. For several models, although they result in successful key recovery with up to 5 000 attack traces, perceived information values are negative, suggesting a sub-optimal profiling model [BDMS22].

Addressing the **ExDL-SCA** questions, we conclude the following:

1. **Where**. Compression usually happens in hidden layers closer to the output layer.
2. **What**. Due to desynchronization effects, the evaluated dataset results in very low SNR. Therefore, we observed fewer bottleneck effects in hidden layers, as some



(a) `conv_1` **(b)** `conv_2` **(c)** `conv_3`

Figure 12: Perceived information values from a CNN trained with the desynchronized `DPAv4.2` dataset.

irrelevant features are still preserved in hidden layers.

3. **Why**. Although the evaluated CNNs can successfully implement a key recovery with less than 5 000 attack traces, noise levels are higher, making the bottleneck less efficient.

7 Possible Use Cases

We suggest several possible use cases for our explainability methodology.

Confirm if a model learns first-order or high-order leakages Recently, the authors of [EST+22] performed a more detailed analysis of the ASCAD databases and found evidence that deep neural networks might learn first-order leakages for some of the target key bytes. This would mean that, contrary to expectation, a deep learning model would not be learning high-order leakages. By applying our explainability methodology, we can quantify the amount of information that a hidden layer contains from secret shares and, as a result, confirm if the model implements a second-order attack.

Evaluate the selected leakage model Some of the publicly available datasets largely considered in previous works (e.g., ASCAD, AES_HD) contain the same key in both profiling and attack sets. For some leakage models, such as $\texttt{S-box}(\texttt{d}_i \oplus \texttt{k}_i)$, the resulting side-channel metrics may indicate successful key recovery even when this is not the case. This situation happens when the model adapts to the key that is set for both profiling and attack sets and not to actual leakages. By applying our explainability approach, it is possible to confirm if a network that generalizes to a fixed key scenario is actually learning existing leakages or simply overfitting.

Tuning layer hyperparameters The perceived information metrics from Eq. (20) quantify the amount of information learned or compressed from input features. In a hyperparameter search, these metrics may help the evaluator to tune some of the hyperparameters to maximize the perceived information. Additionally, with the explainability, it is possible to identify layers with potentially too narrow bottlenecks, which would make the subsequent layer not receive enough information about the leakages. Moreover, our method allows identifying layers that overfit to noise, which degrades the overall performance of the attack. Still, it is an open question whether a modification of one hyperparameter in a single hidden layer affects the learning process of the rest of the network. Therefore, using our methodology to fix a single layer still requires deeper investigation.

Reduce false positives in deep learning-based leakage assessment Moos et al. [MWM21] proposed a deep learning-based leakage assessment (DL-LA) approach, in which a neural network is trained to distinguish between two groups or distributions. DL-LA is also efficient against masked implementations. Therefore, to avoid false positives, our proposed methodology can explain if the network trained to implement leakage assessment is also efficiently learning the secret shares or if the model is, for instance, subject to the class imbalancedness problem or overfitting.

8 Conclusions and Future Works

Masking countermeasures are powerful protections against profiling SCA, especially when higher levels of noise are present in side-channel measurements [BDMS22]. Recent research papers have shown that deep learning techniques are effective in defeating masking and hiding countermeasures (see Sections 1 and 3) and in reducing their protective effects in side-channel traces [WP20]. Therefore, it is of great interest for security engineers and evaluators to understand to what extent the implemented protections are sufficient against different adversaries.

The proposed explainability methodology brings more clarity to the understanding of the effect of masking countermeasures against different deep learning-based profiling attacks. Inspired by the information bottleneck principle [TPB00], and through the lens of perceived information [BHM$^+$19], we provide a method to visualize **what** every hidden network layer learns from high-order leakages and **which one** of them effectively performs the unmasking operation when the high-order leakages are recombined. We applied our methodology to real side-channel measurements and verified how hidden layers successfully implement a bottleneck in order to fit relevant features associated with high-order leakages from the target key byte while compressing irrelevant ones. Furthermore, we provided answers to the main explainability questions in all experimental results scenarios.

Finally, we evaluated the mask share learnability of deep learning models when the desynchronization countermeasures are implemented. The application of our explainability methodology to CNNs trained with desynchronized datasets allows us to understand how convolution layers deal with this type of hiding countermeasure. In particular, we verify how compression in convolution layers plays an important role for the model to be able to generalize and bypass the countermeasures.

This research opens several new research directions for deep learning-based SCA. In future works, we will investigate and quantify compression and generalization phases in profiling attacks. The main goal is to find the optimal trade-off between these two phenomena in deep neural networks. Furthermore, we will investigate how to tune specific neural network hyperparameters to achieve satisfactory compression of noise and irrelevant features, leading to more efficient profiling attacks against more noisy datasets. In particular, we will research a new way to create the best possible bottleneck that can efficiently regularize the model and discard noise by keeping the information from relevant features. Finally, we will consider the explainability methodology to differentiate between a deep learning model that fails to learn existing leakages (which could transmit a false sense of security) from a deep learning model considered for a successful security evaluation that fails against a protected implementation (which result in a correct security estimation).

References

[AFDM17] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[BBR$^+$18] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, R. Devon Hjelm, and Aaron C. Courville. Mutual information neural estimation. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 530–539. PMLR, 2018.

[BDMS22] Olivier Bronchain, François Durvaux, Loïc Masure, and François-Xavier Standaert. Efficient profiled side-channel analysis of masked implementations, extended. *IEEE Trans. Inf. Forensics Secur.*, 17:574–584, 2022.

[BH21] Nadia Burkart and Marco F. Huber. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70:245–317, January 2021.

[BHM$^+$19] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 713–737. Springer, 2019.

[BP21] Vaishak Belle and Ioannis Papantonis. Principles and practice of explainable machine learning. *Frontiers in Big Data*, 4, July 2021.

[BPS$^+$20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptographic Engineering*, 10(2):163–188, 2020.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CLM20] Valence Cristiani, Maxime Lecomte, and Philippe Maurine. Leakage assessment through neural estimation of the mutual information. In *Applied Cryptography and Network Security Workshops - ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA,* *Rome, Italy, October 19-22, 2020, Proceedings*, volume 12418 of *Lecture Notes in Computer Science*, pages 144–162. Springer, 2020.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[CZLG21] Pei Cao, Chi Zhang, Xiangjun Lu, and Dawu Gu. Cross-device profiled side-channel attack with unsupervised domain adaptation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):27–56, 2021.

[dCGRP19] Eloi de Chérisey, Sylvain Guilley, Olivier Rioul, and Pablo Piantanida. Best information is most successful mutual information and success rate in side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):49–79, 2019.

[EST$^+$22] Maximilian Egger, Thomas Schamberger, Lars Tebelmann, Florian Lippert, and Georg Sigl. A second look at the ASCAD databases. In Josep Balasch and Colin O'Flynn, editors, *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*, volume 13211 of *Lecture Notes in Computer Science*, pages 75–99. Springer, 2022.

[FMPR10] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010.

[GP20] Ziv Goldfeld and Yury Polyanskiy. The information bottleneck problem and its applications in machine learning. *CoRR*, abs/2004.14941, 2020.

[GPQ11] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.

[GSC$^+$19] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. Xai&#x2014;explainable artificial intelligence. *Science Robotics*, 4(37):eaay7120, 2019.

[Gut22] Frederico Guth. *The emergence of an information bottleneck theory of deep learning.* Universidade de Brasília (UnB), January 2022.

[HGG19] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Waterloo, ON, Canada, August 12-16, 2019, Revised Selected Papers*, volume 11959 of *Lecture Notes in Computer Science*, pages 645–666. Springer, 2019.

[Hol18] Andreas Holzinger. From machine learning to explainable ai. In *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*, pages 55–66, 2018.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, 1996.

[KPH$^+$19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[LPK20] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1):18, December 2020.

[Mas94] James L. Massey. Guessing and entropy. In *In Proceedings of the 1994 IEEE International Symposium on Information Theory*, page 204, 1994.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 145–167. Springer, 2019.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.

[MOP06] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, `http://www.dpabook.org/`.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[MV20] Ricards Marcinkevics and Julia E. Vogt. Interpretability and explainability: A machine learning zoo mini-tour. *CoRR*, abs/2012.01805, 2020.

[MWM21] Thorben Moos, Felix Wegener, and Amir Moradi. DL-LA: deep learning leakage assessment A modern roadmap for SCA evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):552–598, 2021.

[PBP21] Guilherme Perin, Ileana Buhan, and Stjepan Picek. Learning when to stop: A mutual information approach to prevent overfitting in profiled side-channel analysis. In Shivam Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE*

2021, Lugano, Switzerland, October 25-27, 2021, Proceedings, volume 12910 of *Lecture Notes in Computer Science*, pages 53–81. Springer, 2021.

[PDN22] Jeremy Petch, Shuang Di, and Walter Nelson. Opening the black box: The promise and limitations of explainable machine learning in cardiology. *Canadian Journal of Cardiology*, 38(2):204–213, 2022.

[PPM$^+$21] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. Cryptology ePrint Archive, Paper 2021/1092, 2021. https://eprint.iacr.org/2021/1092.

[PWP21] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Cryptol. ePrint Arch.*, page 1414, 2021.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[RLMI21] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A side journey to titan. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 231–248. USENIX Association, August 2021.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):677–707, 2021.

[SBD$^+$18] Andrew M. Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D. Tracey, and David D. Cox. On the information bottleneck theory of deep learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[SMY09] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 443–461, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[SS17] DJ Strouse and David J. Schwab. The deterministic information bottleneck. *Neural Comput.*, 29(6):1611–1630, 2017.

[ST17] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017.

[TPB00] Naftali Tishby, Fernando C. N. Pereira, and William Bialek. The information bottleneck method. *CoRR*, physics/0004057, 2000.

[vdVPB20] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In Guido Marco Bertoni and Francesco Regazzoni, editors, *Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1-3, 2020, Revised Selected Papers*, volume 12244 of *Lecture Notes in Computer Science*, pages 175–199. Springer, 2020.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):147–168, Jun. 2020.

[WHJ+21] Yoo-Seung Won, Xiaolu Hou, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Back to the basics: Seamless integration of side-channel pre-processing in deep neural networks. *IEEE Trans. Inf. Forensics Secur.*, 16:3215–3227, 2021.

[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):389–415, 2020.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2020/1293, 2020. https://eprint.iacr.org/2020/1293.

[WWJ+21] Lichao Wu, Yoo-Seung Won, Dirmanto Jap, Guilherme Perin, Shivam Bhasin, and Stjepan Picek. Explain some noise: Ablation analysis for deep learning-based physical side-channel analysis. *IACR Cryptol. ePrint Arch.*, page 717, 2021.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.

A Random Hyperparameter Search

Tables 1 and 2 provide the ranges for random search for MLP and CNN architectures, respectively. For each scenario (dataset and architecture), we ran 1 000 hyperparameter search attempts, which was enough to find at least fifty successful models.

Table 1: Hyperparameter search options and ranges for MLPs.

Hyperparameter	**Options**
Optimizer	`Adam`
Dense Layers	2, 3, 4, 5, 6
Neurons	20, 40, 50, 100, 150, 200, 300, 400
Activation Function	`elu`, `selu`, `relu`
Learning Rate	0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001
Batch Size	400
Epochs	100
Weight Initialization	`random_uniform`, `glorot_uniform`, `he_uniform`, `random_normal`, `glorot_normal`, `he_normal`
Regularization	`None`, `l1` or `l2`
`l1` or `l2`	0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001
Total Search Space	**174 960**

Table 2: Hyperparameter search options and ranges for CNNs.

Hyperparameter	**Options**
Optimizer	`Adam`
Dense Layers	1, 2
Convolution Layers	1, 2, 3, 4
Neurons	20, 50, 100, 200
Filters	4 , 8, 12, 16 ($\times$) Conv. Layer Index)
Kernel Size	2, 4, 6, 8, 10, 20, 30, 40
Strides	2, 3, 4, 5, 10, 15, 20
Pooling Size	2
Pooling Stride	2
Activation Function	`elu`, `selu`, `relu`
Learning Rate	0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001
Batch Size	400
Epochs	100
Weight Initialization	`random_uniform`, `glorot_uniform`, `he_uniform`, `random_normal`, `glorot_normal`, `he_normal`
Regularization	`None`
Total Search Space	**903 168**